© 2021 Tianyuan Liu

HARDWARE-ASSISTED PRIVACY ENFORCEMENT IN COMMERCIAL DRONE-AS-A-SERVICE APPLICATIONS

BY

TIANYUAN LIU

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2021

Urbana, Illinois

Doctoral Committee:

Professor Klara Nahrstedt, Chair
Professor Carl A. Gunter
Professor Adam Bates
Doctor Claudiu B. Danilov, Boeing Research & Technology

ABSTRACT

Unmanned Aerial Vehicles (UAVs), also known as drones, have become more popular in commercial activities than before. Many third-party drone service providers offer their drones and pilots to assist client businesses in a variety of missions. Such a business model is also known as Drone-as-a-Service (DaaS) model.

However, the adoption of DaaS applications has been severely impeded due to the potential safety and privacy risks of drones. A malicious drone can fly over residential area and spy on citizens' information. When such drones are equipped with sensors, they can also eavesdrop or devastate private data that goes through wireless sensors. The public damage of the drones are enlarged in DaaS applications because the client often has very limited transparency on the hired drones.

To tackle the above challenges, I present Hardware-Assisted Privacy Enforcement (HAPE) as a potential solution for the privacy issues in DaaS applications. The design of HAPE relies on the hardware-assisted security components, which are installed on the drones, to act as an external source of trust. Therefore, it can be used in various situations such as encrypting sensitive data, authorizing private access, and generating provenance. Based on HAPE, I design three systems to enhance the location privacy, the data privacy and the assignment and management of the DaaS applications. The experiments on these prototypes confirm that HAPE is a viable solution to mitigate the privacy threat of drones in DaaS applications.

To my parents and friends, for their love and support.

ACKNOWLEDGMENTS

In the first place, I want to thank my advisor, Professor Klara Nahrstedt, for her guidance during my Ph.D. study. She has always been patient on me while providing many valuable advice on my research and study. Beyond that, her thoughtful kindness helped me many times when I was stuck on physical or mental problems. I can only make this far to finish my Ph.D. thanks to her support.

Secondly I would like to thank the rest of the thesis committee members, Professor Carl Gunter, Professor Adam Bates and Doctor Claudiu Danilov for their active participation and continuous advising. They have contributed to this thesis either directly or indirectly. It is my pleasure to work with them on these research topics.

In addition, I want to express my sincere gratitude to my parents. My father Rongmu Liu and my mother Meizhen Huang have been supporting me remotely all the time in the past years. Their love and care are always adding to my strength and courage to confront the difficulties.

I would also like to thank the Department of Energy, the National Science Foundation and Boeing Research & Technology for their financial support on my research.

Last but not least, I am grateful to the Monet members and all the friends I met during my Ph.D. time. They have supported me in both of my research and life. A partial list of them includes: Haiming Jin, Hongyang Li, Zhenhuan Gao, Tuo Yu, Tarek Elgamal, Hongpeng Guo, Zhe Yang, Bingzhe Liu, Du Su, Tong Meng.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION 1
1.1 Motivation and Challenges 1
1.2 Thesis Statement 2
1.3 Thesis Overview 2
1.4 Thesis Organization 4

CHAPTER 2 BACKGROUND 5
2.1 Drones and Drone-as-a-Service (DaaS) Application 5
2.2 Trusted Execution Environments 5
2.3 Homomorphic Encryption 6
2.4 Smart Contract 7

CHAPTER 3 RESEARCH OVERVIEW 8
3.1 Enforcing Location Privacy for Commercial DaaS Applications 8
3.2 Enforcing Data Privacy for Commercial DaaS Applications 9
3.3 Enabling Efficient Contract Management for Trustworthy Commercial DaaS Applications 10

CHAPTER 4 RELATED WORK 13
4.1 Drone Applications 13
4.2 Drone Privacy 13
4.3 Trusted Execution Environment 13
4.4 Fully Homomorphic Encryption 14
4.5 Blockchain and Smart Contract 15
4.6 Drone Path Planning 15

CHAPTER 5 ENFORCING LOCATION PRIVACY FOR COMMERCIAL DAAS APPLICATIONS 16
5.1 Preliminaries 16
5.2 AliDrone Protocol 18
5.3 Trustworthy Proof-of-Alibi 20
5.4 System Evaluation 24
5.5 Summary 30

CHAPTER 6 ENFORCING DATA PRIVACY FOR COMMERCIAL DAAS APPLICATIONS 31
6.1 Preliminaries 31
6.2 System Design 35

6.3 Hardware and Implementation 38
6.4 System Evaluation 39
6.5 Summary 41

CHAPTER 7 ENABLING EFFICIENT AND TRUSTWORTHY CONTRACT MANAGEMENT COMMERCIAL DAAS APPLICATIONS 43
7.1 Preliminaries 43
7.2 System Design 47
7.3 Task Assignment Problem 49
7.4 System Evaluation 55
7.5 Summary 57

CHAPTER 8 CONCLUSION AND FUTURE WORK 59
8.1 Summary 59
8.2 Lessons Learned 59
8.3 Future Work 60

REFERENCES 62

CHAPTER 1: INTRODUCTION

In the recent years, the drone technology has enabled many promising applications. In addition to the military purposes, many businesses are paying more attention to the commercial usage of drones. For example, Amazon announced its Air Prime Delivery Service [1] since 2013, aiming to deploy small drones to deliver lightweight packages. Besides, commercial drones are also deployed in applications such as infrastructure construction, precision agriculture, and photography, which were once done by humans [2, 3, 4]. Although the cost of commercial drones has been decreasing over the past decade, it is still a considerable amount of investment for businesses to purchase devices, train operators and build their own drone based applications. Therefore, an industry of drone providers come into play. They provide guidelines to design aerial solutions, lease certificated pilots and drones to collect data, and analyze the collected data for their client. Such a business model is known as Drone-as-a-Service (DaaS) model.

1.1 MOTIVATION AND CHALLENGES

Despite all the benefits of DaaS applications, the public has shown great concern of privacy for drones [5, 6, 7, 8]. For example, a malicious drone operator might use a drone equipped with a high-resolution camera to fly over residential area and spy on citizens' private information. Since 2010, the Federal Aviation Administration (FAA), has been working on the regulations to control the risks of commercial drone usage. The most recent rules [9] include requirements on the pilots, the drone specs, and the locations where drones are allowed to fly. However, these rules mainly focus on the safety protection but fail to enforce privacy compliance in DaaS applications.

The challenges in enforcing privacy compliance in DaaS applications are two-fold. On the one hand, it is hard to detect a drone from the ground due to the small size and fast speed of drones. Unlike ground transportation vehicles, the route taken by a drone is more flexible and unpredictable. Many technologies, e.g., radar surveillance, audio surveillance, video surveillance, radio frequency (RF) surveillance, have been developed for drone detection and localization. However, most of these technologies require expensive external detectors to achieve accurate and real-time detection. For example, the AeroScope drone detection system developed by DJI sells at $340,000 with a $44,000 annual maintenance fee. This is not realistic to be setup by every individual. On the other hand, the authentication of drones is a critical issue. In the package delivery applica-

tion, for example, an attacker drone may impersonate the legitimate one in order to steal packages. Although many software-based solutions were proposed, e.g. using digital certificates to enable publicly verifiable drone identity, such approaches still pose potential risks due to the fact that the attacker, as the owner of the drone, has the privileged access to the device. An attacker may root the software stack in the drone controller and inject malicious code to bypass the software-based defense mechanisms.

In this thesis, I present a general strategy to solve the privacy challenges of DaaS applications by assuming the existence of *hardware-assisted security extensions* on the drones, and hence name this strategy as Hardware-Assisted Privacy Enforcement (HAPE). The hardware-assisted security techniques, i.e. Intel software guard extension (SGX) [10] and ARM TrustZone [11, 12], enable a trusted execution environment (TEE) to secure sensitive code and data even if the system is compromised with root access. Utilizing TEE allows us to build systems with stronger security privileges such as two-factor authentication and peripheral I/O protection [13, 14, 15, 16, 17]. As more and more manufacturers have introduced hardware security modules into low cost system-on-a-chip (SoC), it can be foreseen that hardware security features will be affordable and enabled on every drone controller in the near future. A TEE enabled drone can store and compute sensitive data in the secure enclave and defend against attacks from software stack. It allows DaaS applications to compute trustworthy proofs as a demonstration for privacy compliance.

1.2 THESIS STATEMENT

Hence, I claim that the following statement is true:

Hardware-assisted security techniques must play a critical role in enforcing the privacy compliance in commercial DaaS applications.

1.3 THESIS OVERVIEW

In this thesis, I present several research topics following the HAPE strategy to overcome certain privacy problems. This section provides an overview of each work.

1.3.1 Enforcing Location Privacy for Commercial DaaS Applications

In Chapter 5, we claim that Proof-of-Alibi (PoA) protocols should serve as the basis for enforcing location privacy compliance in DaaS applications. We design and imple-

ment AliDrone, a trustworthy PoA protocol that enables individual drones to prove their non-entrance to certain No-fly-zones (NFZs) to a third party Auditor. AliDrone leverages trusted hardware to produce cryptographically signed GPS readings within a secure enclave, preventing malicious drone providers from being able to forge geolocation information. AliDrone features an adaptive sampling algorithm that reacts to NFZ proximity in order to minimize the processing cost. Through laboratory benchmarks and field studies, we demonstrate that AliDrone provides strong assurance of geolocation while imposing a small overhead on CPU utilization and memory consumption.

1.3.2 Enforcing Data Privacy for Commercial DaaS Applications

In Chapter 6, we present a solution that enables drones to collect and process private client data from remote data sites in a trustworthy and efficient manner. We design and implement the Secure Homomorphic Encryption (SHE) framework. SHE combines trusted hardware enclave and homomorphic encryption technologies to provide strong privacy primitives on client data. SHE features in a recrypt technique such that the computation and communication overhead for homomorphic encryption on the client data is minimized. In addition, SHE takes the advantage of drones' travelling time to run data aggregation tasks in order to speed-up data processing. The laboratory experiments demonstrate that SHE can meet the performance requirement in many common data processing and aggregation missions.

1.3.3 Enabling Efficient Contract Management for Trustworthy Commercial DaaS Applications

In Chapter 7, we present UAVChain, a blockchain-based solution that enables trustworthy and efficient management of drone services. UAVChain is designed with a rich trust-by-default toolbox to establish trust between the clients and drone providers. In addition, it abstracts the interactions in the drone-based applications with several smart contracts, and thus achieves efficient task management. Last but not least, UAVChain also applies a task assignment algorithm to minimize resource utilization to complete the tasks. To validate our design, we implement a proof-of-concept system of UAVChain and present various system benchmarks measured in a controlled laboratory environment. Also, we demonstrate simulations on the real-world datasets that our task assignment algorithm achieves close-to-optimal solution in a short amount of time.

1.4 THESIS ORGANIZATION

The rest of this thesis is organized as follows. Chapter 2 provides the background of the techniques used in this thesis. Chapter 3 presents an overview for each research work, including the problem description, system scope and the high-level workflow. Chapter 4 introduces the existing or related solutions on each research topic. Chapter 5 demonstrates the design of *AliDrone*, the solution to enforce location privacy compliance in DaaS applications. Chapter 6 presents the *Secure Homomorphic Encryption* framework as an enhancement of data privacy protection from the DaaS service providers. Chapter 7 describes *UAVChain* as the trustworthy management framework for DaaS applications. Chapter 8 concludes the insight of this thesis and provides future research directions.

CHAPTER 2: BACKGROUND

In this chapter, we introduce the background and technology that is related to our research.

2.1 DRONES AND DRONE-AS-A-SERVICE (DAAS) APPLICATION

A drone, *a.k.a* unmanned aerial vehicle (UAV), is an aircraft without a human pilot onboard. Such devices can be controlled remotely by the operator within a distance of 2,000 meters. As of 2019, the most popular commercial drones cost from $300 to $3,000. Unlike the entertainment UAVs which can only last for no more than 20 minutes, the commercial drones can fly at the speed of 50 mph with a flight duration of 30-40 minutes.

According to Federal Aviation Administration (FAA) [18] in U.S., one must obtain a Remote Pilot Certificate to be able to operate drones. The certificate requires the pilot to pass an initial aeronautical knowledge test and complete a remote pilot test using the electronic FAA Integrated Airman Certificate and/or Rating Application (IACRA) system.

Due to the high cost of building drone infrastructure and training pilots, many businesses hire drone providers to accomplish their aerial demands. Such business model is referred to as the Drone-as-a-Service (DaaS). Drone providers such as DroneUp, LLC [19] and PrecisionHawk [20] offer DaaS applications in various missions, including infrastructure construction, pipeline monitoring and agriculture.

2.2 TRUSTED EXECUTION ENVIRONMENTS

Trusted Execution Environment (TEE) is a set of hardware-assisted security extensions added to the processors. These processors partition the hardware and software and run a separated subsystem known as "secure world" in addition to the normal operating system, *a.k.a.* "normal world". The TEE technology is programmed into the hardware to protect the memory and peripherals. Consequently, security is enforced without degrading the system performance. TEE can be implemented on commercial secure hardware such as ARM TrustZone [11] and Intel SGX [21].

OP-TEE is an open source project for TEE in Linux using the ARM TrustZone technology. It implements a TEE client in the normal world and a TEE core in the secure

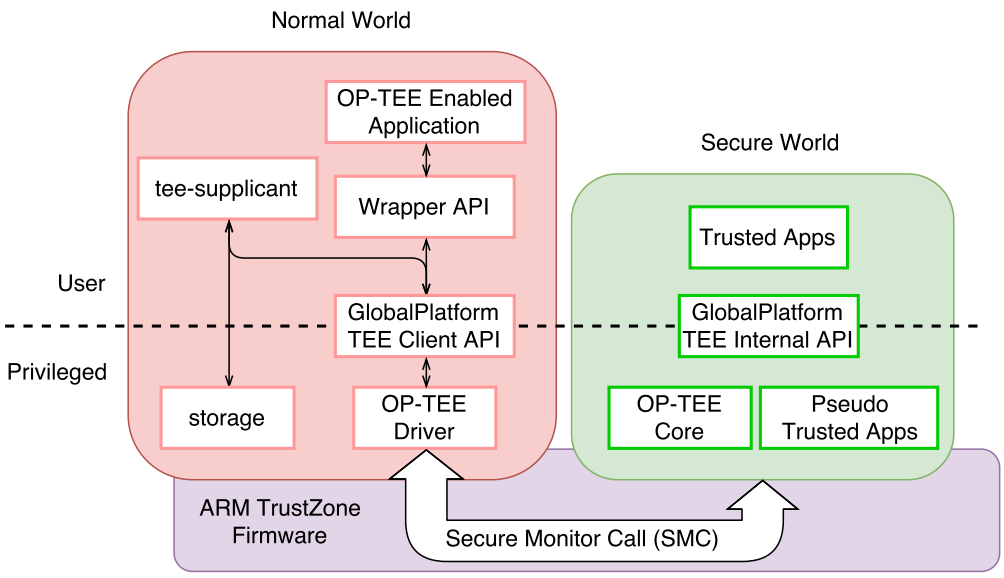


Figure 2.1: OP-TEE Architecture. The code and data in secure world are protected by hardware. The switching between two worlds are triggered via Secure Monitor Call (SMC).

world using the GlobalPlatform TEE System standard. Figure 2.1 shows the architecture of OP-TEE.

OP-TEE provides a minimal secure kernel (OP-TEE core) which runs in parallel with a normal world OS such as Linux. It provides drivers (OP-TEE Driver) for the normal world OS to communicate with the secure world. The transition between the two worlds are done via Secure Monitor Calls (SMC). It uses a daemon service in the normal world, i.e., tee-supplicant, to assist the Trusted OS with the miscellaneous such as storage access.

OP-TEE allows programmers to develop Trusted Applications (TAs) in the User space of the secure world [22]. TAs are signed by a private key which is unknown to the user in the normal world when the secure world is compiled. Every TA is assigned a unique UUID. When an OP-TEE enabled application calls an interface provided by a specific TA, it provides the associated UUID and the interface ID. Then, the tee-supplicant will locate the TA by the UUID in the storage and help the OP-TEE core to load the TA.

2.3 HOMOMORPHIC ENCRYPTION

A homomorphic encryption scheme allows one to perform certain computations on the encrypted data without being able to decrypt it. Fully homomorphic encryption (FHE) provides stronger capability by allowing arbitrary computation on the ciphertext [23].

Conceptually, given a plaintext input x, an FHE encryption key pk and an arbitrary function $f(\cdot)$, it is guaranteed that

$$\text{FHE-Enc}(f(x), \mathsf{pk}) = f(\text{FHE-Enc}(x, \mathsf{pk})). \tag{2.1}$$

FHE thus enables various applications to outsource private computation [24, 25, 26]. For instance, in the context of cloud computation, a client may send the FHE encrypted data to a remote server. The server can then perform homomorphic computation on the ciphertext and return the encrypted result to the client. Finally, the client can use the FHE decryption key to recover the result. The protection of user data privacy is achieved since the server is unable to decrypt the user data.

However, FHE encryption consumes a large amount of computation resource. Also, the FHE ciphertext is thousands of times larger than the plaintext message such that transmitting FHE ciphertext may result in significant communication overhead. The SHE framework leverages TEE to avoid the computation and communication overhead and thus produce a more efficient and feasible solution for using FHE in third-party UAV services.

2.4 SMART CONTRACT

A smart contract is a collection of code and data embedded in the blockchain. Smart contracts are widely applied for the first time with the implementation of Ethereum blockchain and Solidity language [27]. The deployment of a smart contract is done by sending a transaction that includes the compiled smart contract code to a special receiver address. The initialization code will be executed when this transaction is added to the blockchain. The security of the contract is guaranteed by the consensus protocol from the blockchain.

Ethereum is a decentralized, open-source blockchain featuring smart contract functionality. Ether is the native cryptocurrency of the platform. The Ethereum Virtual Machine (EVM) is capable of running smart contract code. Solidity is one of the most popular programing language for the EVM-based smart contract. For example, it can be used to create smart contract applications such as voting, crowdfunding, blind auctions and multi-signature wallets.

CHAPTER 3: RESEARCH OVERVIEW

In this chapter, we present an overview of our research that has already been done, including the enforcement of location privacy, data privacy, and the trustworthy proof-of-work in DaaS applications.

3.1 ENFORCING LOCATION PRIVACY FOR COMMERCIAL DAAS APPLICATIONS

One promising countermeasure for mitigating drone surveillance is the establishment of no-fly-zones (NFZs) over privacy-sensitive locations. If a drone is sufficiently far away from a sensitive area, surveillance cannot be carried out successfully. The FAA has designated a variety of NFZs, primarily for safety purposes, around critical infrastructures such as airports. An established NFZ specifies that no drone is permitted to fly within 5 miles of the protected location. To more effectively notify the pilots of NFZs in their area, the FAA has even published the B4UFLY mobile app [28]. Unfortunately, regulation alone cannot prevent drones from flying over restricted areas; as the drone navigates in open airspace, it is hard for an observer on the ground to accurately determine the location of a drone. Instead, what is needed are a reliable means of tracking drone locations for the detection of NFZ policy violations.

We present the design and implementation of *AliDrone*, a geo-location based alibi protocol that enables drones to generate proof-of-non-entrance to an NFZ. We define three roles in the system: *Zone Owners* that own some property, *Drone Provider* that hires a pilot who controls the drone and navigates it through an area, and *Auditor*, an authorized third party (e.g. local agent of the FAA) that attests drones' locations and detects any non-compliance with NFZ regulations. Before flying, the Drone Provider queries the Auditor for the location of nearby NFZs. While flying, the drone computes an *alibi*, i.e. a signed GPS trace, based on its real-time location. At the end of the flight, the Drone Provider submits the drone's Proof-of-Alibi (PoA) to the Auditor. The Auditor then verifies the PoA and initiates punishment on the Drone Provider if a privacy violation is detected.

We design AliDrone with the consideration that a *dishonest drone provider* may try to navigate the drone over a restricted area without being detected by the Auditor. Such an attacker could attempt to forge an innocent compliant route and compute its alibi based on this forged GPS trace. As a result, an adversary may take a shortcut route or gain pictures of the restricted area. Defending against such adversary is challenging. As the owner of the drone, the Dishonest Drone Provider has privileged access to the drone

software stack as well as any exposed hardware, meaning the attacker could attempt to extract security keys used in the alibi protocol or replace the system components with malicious software.

We demonstrate the hardware-assisted security technique for location privacy and the system design of AliDrone and address the aforementioned challenges in Chapter 5.

3.2 ENFORCING DATA PRIVACY FOR COMMERCIAL DAAS APPLICATIONS

One drawback of DaaS business model is that the clients often have very limited control on the drones. Although flying logs and reports can be provided as the proof of work, the clients cannot fully trust the third-party drone providers on how these logs are captured. Moreover, the state-of-the-art drone setup does not provide privacy guarantee when drones need to interact with data owned by other entities during the mission. A malicious third-party drone provider may abuse the user data without being noticed by the client and data owners.

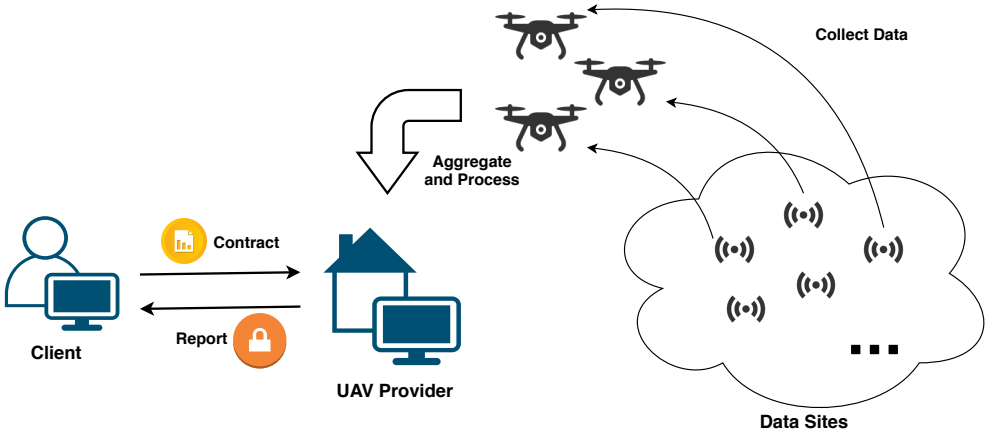


Figure 3.1: The high-level concepts of SHE framework.

To enforce the data privacy compliance, we present the design and implementation of Secure Homomorphic Encryption (SHE), a privacy-preserving data collection and processing framework that enables drones to interact with private user data. We identify three entities in our application scenario: a *Drone Provider* , a *Client*, and multiple *Data Sites*. From the high level as shown in Figure 3.1, the client makes a contract with the drone provider who owns a set of UAVs and certified pilots. The contract requires the drone provider to collect data from a set of data sites and to execute certain data processing algorithm within a given expiration time. We assume that such data sites have

fixed infrastructure sensors (e.g. building sensors, smart city Array of Things, etc.) or Internet-of-Things (IoT) devices owned either by the client or by different parties. Thus, the drone provider must have controlled access to the data. When the UAVs finish the data collection, the drone provider will process the data and return a report to the client.

SHE is designed with the consideration that a dishonest drone provider shall not gain knowledge of the private data during the contract. Homomorphic encryption [29] is thus a good candidate in this situation. However, traditional homomorphic encryption technology consumes extraordinary amount of computation and communication resource on the data sites. Therefore, it cannot be applied on resource limited devices such as sensors or lightweight IoT devices.

We successfully overcome these issues and present SHE as the hardware-assisted security technique for data privacy on drones in Chapter 6.

3.3 ENABLING EFFICIENT CONTRACT MANAGEMENT FOR TRUSTWORTHY COMMERCIAL DAAS APPLICATIONS

Negotiating over a contract with the drone provider is not a fast and easy matter. To deploy a DaaS application, the client company usually first consult the drone provider on its demands. The drone provider will then do the research about the best hardware and software platform for the specific targeted industry and designs a solution that best fits the demands. Therefore, commercial DaaS application today is often provided as a long-term and sustained contract. As a result, small businesses and individual users can hardly benefit from the current DaaS applications. Very often, such users only need a small number of drones to execute some simple tasks for limited runs. It is difficult for these users to find a provider only wishing to use the service for a short period.

To enable efficient contract management, we present the design and implementation of UAVChain, a trustworthy and distributed platform for the management of DaaS applications. UAVChain features in the synergy of the TrustZone technology and a set of smart contracts. The task information, report and payment are managed and stored by an Ethereum-based blockchain. Supported by the public smart contract interfaces, the users can efficiently find a proper provider to execute their aerial demands.

We describe the general workflow of the UAVChain framework in Figure 3.2 using a "parcel delivery" example. Assume we have a *client* Alice who wishes to deliver a parcel from location X to location Y. Alice first posts the task information, including the weight and size of the parcel, the location coordinates of X and Y, and the reward of the task, to the platform. The platform then runs a matching algorithm and assigns a proper *Drone*

Provider, Bob, for this task. After that, Bob sends his drone into the *Task Region* and starts executing the task. When the parcel is dropped at location Y, Bob takes a picture of the delivered parcel, and make a GPS record. The picture and GPS record will be submitted to the platform as the proof of work, and the proof will be verified by Alice. If Alice is happy with the service, Bob will receive the promised reward. Otherwise, an authorized *Mediator* will step in to mediate between Alice and Bob.

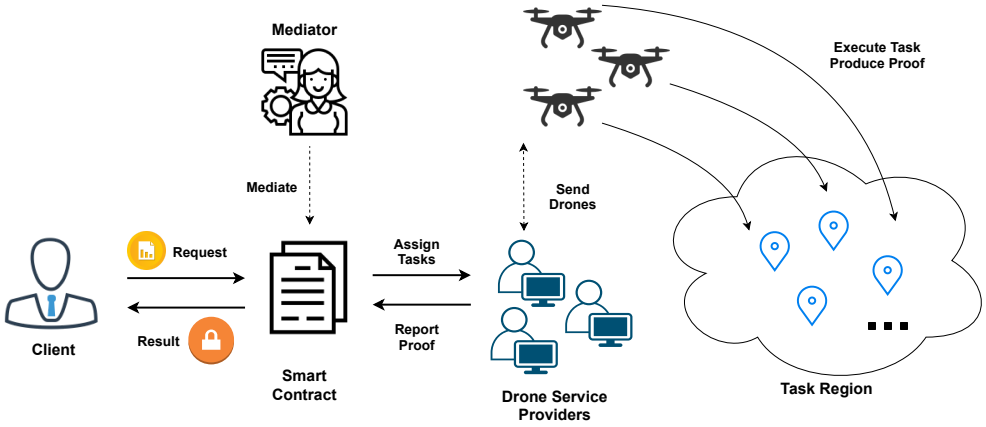


Figure 3.2: The high-level concepts of UAVChain.

From the high level, UAVChain is designed with the guarantee that a dishonest participant cannot bypass the verification stage for the proof of work. We consider both clients and drone providers as the potential adversaries. A malicious drone provider may try to skip executing the task by submitting a forged report or proof. A dishonest client may try to avoid paying the reward by negating the proof. In addition, we also consider the attacks from the outsiders who pretend to be a legitimate participant and try to crash the system by infinitely triggering unnecessary computation flows. The idea of shutting down these attacks is done through the economy mechanism built in the smart contracts. The smart contracts require each participant to deposit some amount of cryptocurrency at certain steps, and do not allow them to withdraw until the end of the verification. As a result, it is guaranteed that an honest participant will get or pay the expected amount of reward. Otherwise, the detected adversary will lose the deposit.

UAVChain relies on hardware-assisted security mechanics from the drones to generate verifiable and unforgeable proof. Similar to the methods presented in Section 3.1 and 3.2, UAVChain allows the clients to specify the demanded types of proof to be submitted by the drone providers. Hence, it is compatible with various types of proofs including location, time, encrypted data and images.

In addition, UAVChain uses a task assignment algorithm to find a proper drone provider

for the client. The best match of a drone provider should (1) meet the requirement of the task description, and (2) can complete the task using a minimal number of drones. However, the computation of the navigation time is reduced to the Travelling Salesman Problem (TSP) and thus it is NP-hard. Therefore, we use linear programming to formulate the problem and design a heuristic algorithm to solve it.

Therefore, UAVChain can serve as the management platform for trustworthy DaaS applications. We will present it in more details in Chapter 7.

CHAPTER 4: RELATED WORK

In this chapter, we introduce the past literature related to this thesis. An overview of the related work is shown in Figure 4.1.

4.1 DRONE APPLICATIONS

The drones have enabled many promising commercial applications such as package delivery, infrastructure construction, precision agriculture, and photography [1, 2, 3, 4]. In the academia, many researchers are also interested in using this flying platform in varies activities including wifi relay, data collection, remote surveillance and video processing [30, 31, 32, 33, 34].

4.2 DRONE PRIVACY

Privacy is one of the major concerns about the pervasive deployment of drones. Nevertheless, among cybersecurity, privacy and public safety issues [35], the previous research on drone privacy was limited to regulations [5, 36, 37]. The most promising approach suggested by Cavoukian [38] was to apply Privacy by Design (PbD) principle to drone technologies. However, even though the drone system is complied with PbD, an authorized drone operator can always attach a small camera to the drone and covertly capture surveillance videos.

Recent research suggests that geofencing is an effective system to prevent drones from restricted areas [39, 40, 41, 42]. However, since accurate geofencing devices are often expensive, it is unrealistic for all the citizens to deploy this technology in their properties.

4.3 TRUSTED EXECUTION ENVIRONMENT

Trusted Execution Environment (TEE) technology has received excessive research interest in recent years. Intel software guard extension (SGX) [10], ARM TrustZone [11, 12] and virtual TEE solution [43] made it possible to secure sensitive code or data even if the system is compromised with root access.

Utilizing TEE allows us to build systems with stronger security privileges [13, 14]. Specifically, [15] designed an efficient two-factor authentication scheme on ARM TrustZone and achieved comparable security assurance to hardware token based solution. Liu

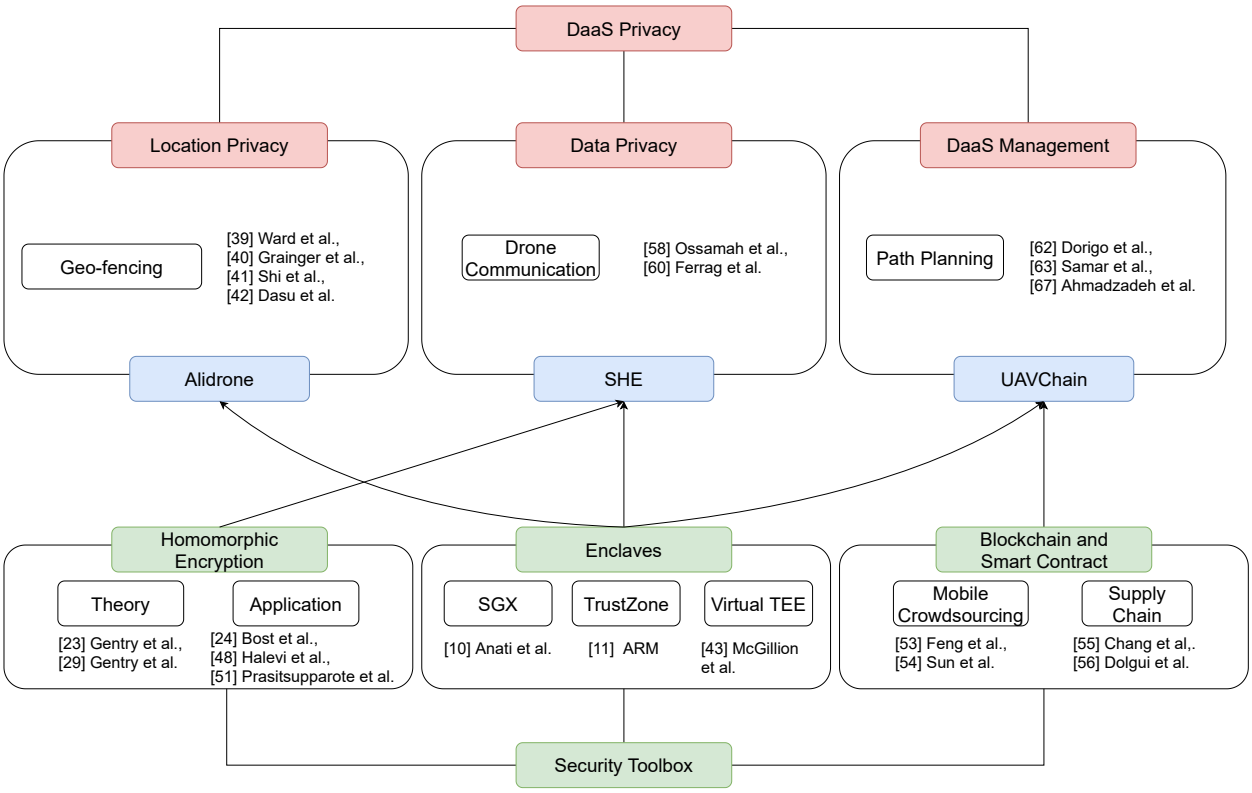


Figure 4.1: An overview of the related work.

and Srivastava [16] used ARM TrustZone to protect essential peripherals for the UAVs. [44] presented a proof-of-alibi procotol based on TrustZone to verity geo-location data of UAVs. Hasan and Mohan [17] deployed TEE to secure IoT devices against the control spoofing attack with the guarantee of timing requirement.

4.4 FULLY HOMOMORPHIC ENCRYPTION

Fully homomorphic ecnryption (FHE) allows arbitrary operations to be executed on ciphertext. Since Gentry [29] first introduced a feasible FHE construction, a large number of mathematical constructions and algorithmic concepts have been proposed to improve the computation and memory efficientcy of FHE [23, 45, 46, 47].

Meanwhile, a number of FHE libraries were developed and open-sourced [48, 49, 50]. These libraries allowed FHE to be experimented on various real-world applications. Previous literature demonstrated that comprehensive algorithms such as classification and machine learning can be implemented with FHE using proper approximation [24, 25, 26]. FHE is also proved to be feasible in the cheaper devices (e.g. IoT devices, wearable sensors) which only have limited computation resources [51, 52].

4.5 BLOCKCHAIN AND SMART CONTRACT

Blockchain and smart contract are secured technologies as they by default feature in private key cryptography and peer-to-peer network. Although we are the first that apply these technologies in the management of drone services, similar approaches have been widely studied in related topics. [53] and [54] demonstrate blockchain-based machanisms to enable trustworthy and privacy-preserving features in mobile crowdsourcing. [55, 56, 57] propose several smart contract designs such that the supply chain products can be tracked efficiently.

Beyond the management of tasks, the blockchain technology also attracts a lot of interest in securing the drone systems. It has been pointed out that blockchain may offer a solution to many security challenges faced by the drone industry [58]. [59] and [60] use blockchain to secure the communication signals for the drones. [61] designs an intrusion detection system based on blockchain for the drone delivery services.

4.6 DRONE PATH PLANNING

Drone path planning problem has attracted widespread attention with the technical mutuality of drones. The essence of path planning is to solve the traveling salesman problem [62], which is NP-hard. Different algorithms were proposed towards the drone path planning problem [63, 64, 65, 66]. In terms of algorithm formation, most of the proposed approaches can be divided into two categories, optimal algorithms and heuristic algorithm.

The optimal algorithms are usually formulated into mathematical programming problems to obtain the exact optimal solution [67]. These mathematical programmings are usually solved using numerical methods such as parameter optimization and simplex method. The heuristic algorithms is more efficient and scalable comparing with the optimal algorithms. Several heuristic drone path planning algorithms includes Dijkstra Algorithm [68] and Floyd Algorithm [69]. However, most of aforementioned algorithms only focus on path planning for a single drone, which can not suffice our needs to plan flying trajectories for multiple drones to cover a large amount of data sites collectively.

CHAPTER 5: ENFORCING LOCATION PRIVACY FOR COMMERCIAL DAAS APPLICATIONS

To enforce location privacy in DaaS systems, we present our design of AliDrone, a proof-of-alibi protocol in this chapter.

5.1 PRELIMINARIES

Model	$S\,(x, y, t)$	=	GPS sample with longitude, latitude and timestamp.
	$z\,(x, y, r)$	=	a circular no-fly-zone with longitude, latitude and radius.
	id_{drone}	=	Identifier of the drone.
	id_{zone}	=	Identifier of a no-fly-zone.
Keys	(T^+, T^-)	=	Asymmetric sign key pair of the Drone TEE.
	(D^+, D^-)	=	Asymmetric sign key pair of the drone.

Table 5.1: Table of Notations Used in This Chapter

5.1.1 Overview

We consider a *Drone Provider* that instructs a drone to navigate a given flight pattern. We represent the drone's activity as a series of samples $S = (lat, lon, t)$, each represented as a tuple of latitude, longitude and timestamp that are sampled from a GPS receiver. A particular drone flight pattern F can thus be summarized as $F = \{S_0, S_1, \ldots, S_n\}$.

This work considers a situation where a drone must navigate an area in which many No-fly-zones(NFZs) are present. We assume all NFZs to be circular, and are defined by $z = (lat, lon, r)$, where lat and lon are the latitude and longitude of the center, and r is the radius of the circle. We refer to the entities who own the NFZs as No-fly-zone owners (or *Zone Owners*). If a drone passes into an NFZ, we say that the privacy of this Zone Owner is violated.

We assume that each drone is associated with an identifier, similar to a vehicle license plate, which is visible by an observer on the ground. If a Zone Owner spots a drone close to her NFZ, she may suspect that privacy violation has occurred. The Zone Owner will record the drone ID and report the incident to an *Auditor*, which is an authorized third party, e.g., a local Federal Aviation Administration (FAA) agent. The Auditor uses the drone ID to recover the flight pattern F from the Drone Provider, then determines if the

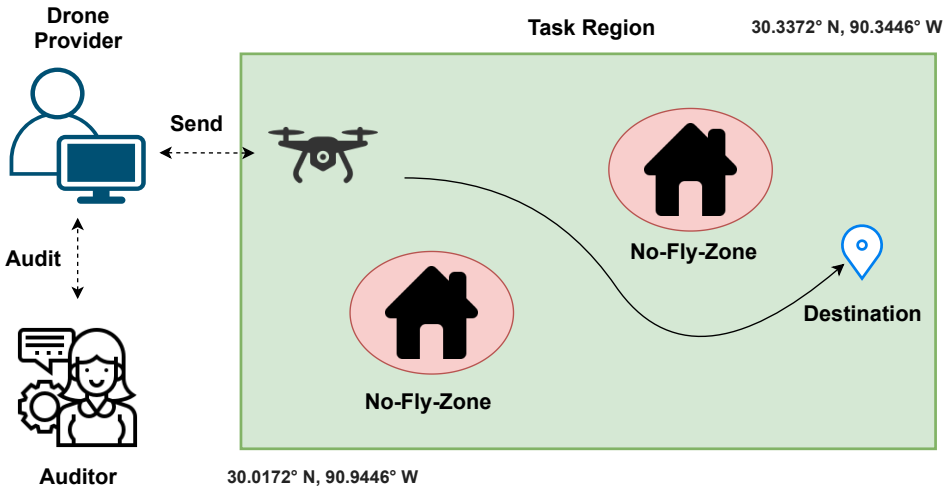


Figure 5.1: An overview of system workflow. The process starts where the Zone Owner submits the coordinates to the Auditor (task 1). Then, Drone Provider submits its flight plan to the Auditor and in response receives the NFZs within the flight zone (task 2 and 3). After the flight, Drone Provider provides the proof-of-alibi, showing that its drone has not flown over the restricted NFZs to the Auditor (task 4).

privacy violation did occur. *In our model, the burden of proof rests on the Drone Providers to prove conclusively that their drones could not have been present in the NFZs.*

Different from the traditional solution which attempts to build a virtual boundary around the NFZs using Geo-fencing technologies [39, 40, 41, 42], our design does not require the Zone Owners to install external detecting devices such as radio transmitters or beacon detectors. Our solution only relies on the existing secure hardware to provide a trusted execution environment for drones to generate their Proofs-of-Alibi (PoA). PoA serves to prove that the drone does not enter any of the NFZs on the map during the navigation. If F is insufficient to produce such a PoA, the Auditor concludes that a privacy violation has occurred. The Auditor will then initiate punitive measures against the Drone Provider. The punishment for privacy violation is orthogonal to the purpose of this work, and can be specified through policy or legislation.

5.1.2 Threat Model

We consider the adversary as a dishonest Drone Provider (or rogue drone) that wants to violate NFZ airspace without being detected by the Auditor. Such an adversary may be small business looking to reduce costs by taking a shortcut, or a journalist or amateur Provider attempting to acquire footage from a restricted area [70]. To avoid detection,

the adversary will attempt to forge an innocuous route to present to the Auditor in place of its actual illicit GPS trace. This feat may be attempted through pre-computing a route that does not intersect any NFZ, replaying a previously reported route, or relaying a route from another drone. We use the term *GPS forgery attack* to denote this attack in the rest of this chapter.

We assume the presence of secure hardware within the drone that provides a trusted execution environment (i.e., ARM Trustzone, Intel SGX). Furthermore, we assume that an asymmetric sign key pair is generated within TEE by the hardware manufacturer, and the private key is not known by the Drone Provider. Side channel attacks on the enclaves [71, 72, 73] are not considered in this work.

While the attacker can attempt to install malicious software on the drone platform, we assume the correctness of the GPS hardware. We also do not consider GPS spoofing attacks in which the GPS receiver is manipulated from the ground through the broadcast of incorrect GPS signals [74, 75]; such attacks can be mitigated through existing defenses [76, 77, 78, 79].

5.1.3 Design Objectives

The goal of AliDrone is to protect the privacy of Zone Owners by allowing them to request NFZs upon their properties. The solution enables the drones to present trustworthy, location-based PoAs proving that the drones do not fly over the NFZs. The PoA is verified by a trusted third party, described as the *Auditor* in the previous context. We list our design goals as follows:

Completeness The PoAs generated by the drone must prove that it does not fly over *any* NFZ during the *entire* flight period.

Low Overhead The computation of trustworthy PoAs should impose *small* processing overhead for the drones.

Unforgeability The Auditor *must not accept* any PoA if it is forged by Drone Provider.

5.2 ALIDRONE PROTOCOL

We describe the high level concepts of AliDrone protocol in this section. Three entities are involved in the protocol: a Drone Provider, a Zone Owner and an Auditor. Figure 5.1

demonstrates the interactions among these entities. A summary of cryptographic keys and data used by the protocol is presented in Table 5.2.

Notation	Description	Knowledge
id_{drone}	Identifier of drone. It is visible on the drone.	All parties
id_{zone}	Identifier of NFZ.	All parties
T^-	Private TEE sign key.	Drone TEE
T^+	Public TEE verification key.	Drone Provider/Auditor
D^-	Private sign key of a Drone Provider.	Drone Provider
D^+	Public verification key of a Drone Provider.	Auditor

Table 5.2: Notations of keys and data used by AliDrone protocol. Column **Knowledge** indicates the parties who have access to the information.

Step 0. Drone Registration: We require that a drone should be registered at the Auditor before operated in the field. The Drone Provider generates an asymmetric keypair $\mathcal{D}$ =(D^+, D^-) and provide the public key D^+ to the Auditor. To enable trustworthy report of geo-locations, we require that an asymmetric keypair for the Trusted Execution Envirionment (TEE) on the drone $\mathcal{T}$ = (T^+, T^-) is generated at manufacturing time. The TEE sign key T^- is only accessible by TEE and the verification key T^+ is known to the drone owner when the device is merchandised. At registration, the TEE verification key T^+ should also be submitted to the Auditor. An identifier id_{drone} is then issued to the drone. This identifier is similar to a vehicle license plate, which must be carried on the drone when it operates. Therefore, an entry of registered drone can be expressed as $(id_{\text{drone}},\ D^+,\ T^+)$.

Step 1. Zone Registration: In order to register an NFZ, a Zone Owner submits to the Auditor the coordinates and radius of the property, i.e., $z = (lat, lon, r)$, as well as a proof of ownership. Upon request approval, the Auditor issues an identifier id_{zone} to the Zone Owner and adds a new entry (id_{zone}, z) to the NFZ database.

Step 2-3. Zone Query/Response: Before a drone starts navigation, the Drone Provider should query the auditor for the NFZ information. The query is comprised of the drone id, two GPS coordinates (x_1, y_1) and (x_2, y_2), indicating a rectangular navigation area, and a random nonce signed by the drone sign key D^-, i.e.,

$$(id_{\text{drone}},\ (x_1, y_1),\ (x_2, y_2),\ \text{nonce},\ \text{Sig}(\text{nonce}, D^-)). \tag{5.1}$$

The Auditor first checks if the query is sent from a registered drone by verifying the signature on the nonce. Then, it pulls a list of NFZs $\{z_1, z_2, \cdots, z_m\}$ within the rectangle and responses with the coordinates and radii of the zones. The drone can use the NFZ

information to compute a viable route to its destination.

Step 4. Proof-of-Alibi Submission: During the flight, the drone computes the Proof-of-Alibis (PoAs) and persists the PoAs to the storage. The purpose of the PoA is to show that the drone does not enter any NFZ during the flight. The detailed design of PoA is presented in section 5.3. At the end of the flight, the Drone Provider must submit the PoAs to the Auditor for verification. To enable real-time auditing, the drone could alternately transmit its PoAs in real-time to the Auditor; however, we do not pursue this solution in our work as it would increase battery drain, violating Goal Low Overhead.

5.3 TRUSTWORTHY PROOF-OF-ALIBI

In this section, we introduce the concept and design of Proof-of-Alibi (PoA), which enables drones to generate unforgeable GPS traces. We first explain how the geo-location information serves as a proof of privacy compliance (Completeness goal). Then, we demonstrate an extension in the trusted execution environment (Unforgeability goal) and an optimization to reduce processing overhead (Low Overhead goal).

5.3.1 Possible Traveling Range

To prove that a drone does not enter an NFZ, we show that it is physically impossible to travel into the zones based on its geo-locations. The idea of this proof is based on the fact that drones have a maximum traveling speed $v_{\max}$, which is restricted to 100 mph by the FAA regulation [9]. This enables the computation of the *possible traveling range* using two GPS coordinates.

Consider that the drone produces two GPS samples $S_1 = (x_1, y_1, t_1)$ and $S_2 = (x_2, y_2, t_2)$. Denote the location of the drone at arbitrary time $t \in [t_1, t_2]$ as (x, y), the possible traveling range can be described as an ellipse $\mathcal{E}$ with (x_1, y_1) and (x_2, y_2) being the two focuses: $\mathcal{E}(S_1, S_2) = \{(x, y) \mid d_1 + d_2 \leq v_{\max}(t_2 - t_1)\}$, where $d_i = \sqrt{(x - x_i)^2 + (y - y_i)^2}$.

Suppose the drone operates near an NFZ $z = (x_0, y_0, r_0)$. The GPS samples (S_1, S_2) can prove that the drone does not enter zone z during (t_1, t_2) if the ellipse does not intersect with the circle representing zone z. Otherwise, it suggests that the drone may travel into zone z during $[t_1, t_2]$.

During the flight, we require the drone to collect a set of GPS samples and define the set of the samples as *alibi* $\coloneqq \{S_0, S_1, \cdots, S_n\}$.

Given a set of NFZs $Z = \{z_1, z_2, \cdots, z_m\}$, we say that the alibi is *sufficient* if every pair of two consecutive GPS samples proves impossibility of traveling into all the NFZs, i.e.,

$$\mathcal{E}(S_i, S_{i+1}) \cap \left(\bigcup_{z \in Z} z \right) = \varnothing, \ \forall\, i < n. \tag{5.2}$$

Otherwise, we say the alibi is *insufficient*. Insufficient alibi suggests that the drone may travel into NFZs during the flight. Hence, it does not show compliance with the no-fly rule. If we consider a simple case where only one NFZ is on the map shown in Figure 5.2, the minimum sampling rate that produces sufficient alibi should results in an ellipse that is tangent to the NFZ.

5.3.2 TEE Enabled GPS Sampling

To ensure that such alibi cannot be forged by Drone Providers, our solution leverages trusted hardware to authenticate the GPS data in a Trusted Execution Environment (TEE). We move the sampling logic to the secure world to guarantee that the GPS data is collected from the GPS hardware. The GPS data is signed by the TEE sign key T^- before it leaves the secure world. We define the *Proof-of-Alibi (PoA)* as a series of GPS samples along with the TEE signatures, i.e.,

$$\text{PoA} \coloneqq \{(S_0, \mathsf{Sig}(S_0, T^-)), (S_1, \mathsf{Sig}(S_1, T^-)), \cdots\}. \tag{5.3}$$

The sign key T^- is only available to TEE such that a Drone Provider in the untrusted environment cannot forge the signatures. The verification key T^+ is known to the Auditor at registration stage, and thus the Auditor is able to detect if the GPS data is modified. Our design can be generalized to trusted hardware platforms including Intel SGX and ARM TrustZone. We present the an ARM TrustZone based architecture of AliDrone in Figure 5.3.

The Auditor runs an AliDrone Server. It stores the information of registered drones and NFZs, and provides an interface to query the NFZ information to the drone client. Upon receiving the PoAs from drones, it verifies the sufficiency of the PoAs (see equation (5.2)). After the PoA verification, the AliDrone Server should save the PoAs for a couple of days. This is because a Zone Owner may report a violation afterwards and the PoAs serve as evidence for the accusation.

The drone client consists of three components: GPS Driver, GPS Sampler and Adapter.

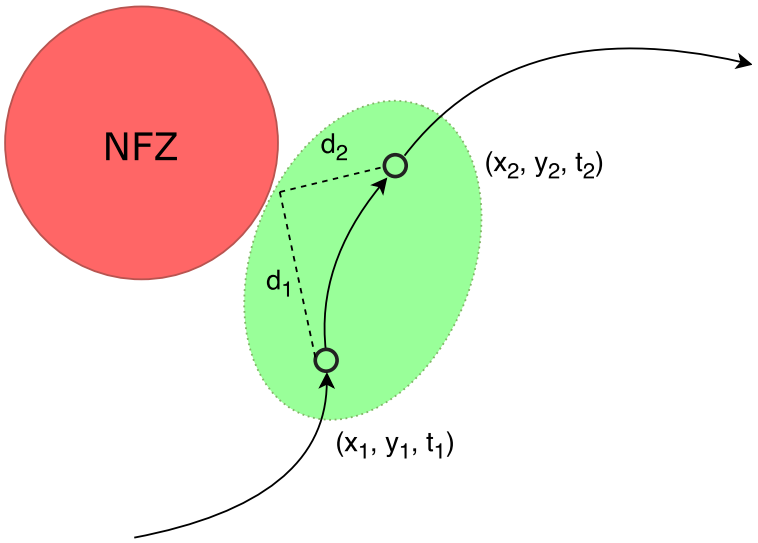


Figure 5.2: Possible traveling range and a single NFZ. The possible traveling range should not intersect with the NFZ to produce sufficient alibi.

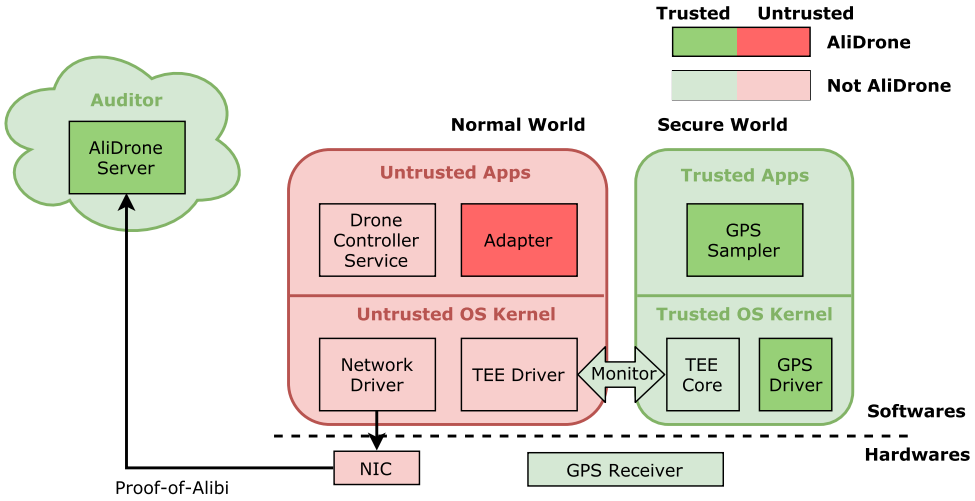


Figure 5.3: AliDrone System Architecture. AliDrone enables trustworthy PoA generation on the drone by performing GPS sampling in a TEE. The GPS data is sampled, encrypted and signed by the trusted application GPS Sampler. The Adapter runs adaptive sampling algorithm and adjusts GPS sampling rate in real time. The Auditor runs AliDrone Server to verify the PoA uploaded by the drone.

GPS Driver runs in the kernel space of the secure world. It is used to access the GPS receiver and parse the raw GPS data into coordinates and timestamps.

GPS Sampler runs in non-privileged mode in the secure world. It exposes an interface `GetGPSAuth` to the Adapter to produce an authenticated GPS sample. It reads the parsed GPS data from the underlying GPS Driver and signs the data with the TEE sign key T^-.

The Adapter is a daemon service in the normal world. It has access to the GPS receiver and controls the PoA sampling rate using the adaptive sampling mechanism, which will be introduced in section 5.3.3. In addition, it is responsible for encrypting the PoA with the public encryption key of the AliDrone Server.

5.3.3 Adaptive Sampling

A commercial GPS receiver can update the GPS measurements with a maximum rate of 5Hz. However, performing frequent sampling in AliDrone is expensive because signature and world-switching operations are costly. Maintaining the maximum sampling rate has a non-negligible amount of processing overhead on the resource limited hardware. Therefore, an adaptive sampling mechanism is essential to minimize the processing overhead for the drones.

As mentioned in section 5.3.1, two samples (S_1, S_2) are sufficient to prove alibi from zone z if the ellipse of possible traveling range does not intersect the zone, i.e., $\mathcal{E}(S_1, S_2) \cap z = \varnothing$.

Given a traveling trace described by a series of samples $\{S_0, S_1, \cdots, S_n\}$ such that $t_i < t_{i+1}$, we can conclude that $\mathcal{E}(S_i, S_j) \subset \mathcal{E}(S_i, S_k),\ \forall\ i < j < k$.

This implies that if the sample pair (S_i, S_k) is sufficient, all the intermediate samples in between are not needed in the PoA. Denote the PoA as a set of samples selected from the trace $\{S_{k_0}, S_{k_1}, \cdots, S_{k_m}\}$ and let the first sample from PoA be $S_{k_0} = S_0$. The task of the Adapter is to find

$$k_{i+1} = \arg\max_j \left(\mathcal{E}(S_{k_i}, S_j) \cap z = \varnothing\right),\ \forall\ k_i < j < n. \tag{5.4}$$

Since the Sampler only samples the current GPS information by demand, it can be too late to recover a previous sample when the current location already violates PoA sufficiency. Therefore, the Adapter must take a sample when the boundaries of the possible traveling range and the NFZ are close.

Consider the worst case where the drone flies towards the NFZ $z = (x_0, y_0, r_0)$ at maximum speed $v_{\max}$. Assume that the GPS receiver has a maximum update rate of R Hz. Let

the last sample recorded in PoA be $S_1 = (x_1, y_1, t_1)$ and the latest sample measured by the Adapter be $S_2 = (x_2, y_2, t_2)$ such that

$$D_1 + D_2 \geq v_{\max}(t_2 - t_1) \tag{5.5}$$

where $D_i = \sqrt{(x_i - x_0)^2 + (y_i - y_0)^2} - r$ is the distance between the drone and the boundary of z. The next GPS update will be made in $\Delta t = \frac{1}{R}$ and the difference of such distance will be $\Delta D = -v_{\max}/R$.

The sample S_2 should be made if the next measurement will be *insufficient*, i.e., $D_1 + D_2 + \Delta D < v_{\max}(t_2 - t_1 + \Delta t)$. Therefore we have

$$D_1 + D_2 < v_{\max}(t_2 - t_1 + 2/R) \tag{5.6}$$

Now we can conclude that a sample should be recorded in PoA if conditions (5.5) and (5.6) are both true.

When multiple NFZs are present, we only need to prove PoA sufficiency for the closest zone. We present the Adaptive Sampling algorithm in Algorithm 5.1. In each iteration, the Adapter first samples the GPS data in the normal world by calling `ReadGPS()` with the same rate R that the GPS receiver updates the measurements. Then, it finds the closest zone from NFZ list. If both conditions (5.5) and (5.6) hold, it calls `GetGPSAuth()`, which acquires the sample and the signature from the GPS Sampler in the secure world.

5.4 SYSTEM EVALUATION

5.4.1 Field Studies

In this section, we evaluate the AliDrone in two cases, each representing a specific pattern of the surrounding no-fly-zones.

Experimental Setup We implement a proof-of-concept prototype on Raspberry Pi 3 and emulate the flight pattern of a drone by driving a vehicle carrying our prototype. As the personal properties may be reserved as NFZs, it is reasonable to assume that the airspace upon roads and public areas like parks are available for commercial drone navigation. We emulate the GPS sampling of drones by driving the vehicle around a small county region. The maximum sampling rate of the GPS sensor was set to 5 Hz and the entire GPS traces including latitude, longitude and timestamps were recorded. The collected

Algorithm 5.1: Adaptive Sampling Algorithm. The adaptation is achieved by skipping unnecessary calls of `GetGPSAuth()` interface.

NextSample $(R,\ S_1,\ Z)$;
Input : R - GPS Update Rate; S_1 - Last GPS Sample in PoA; Z - NFZ list.
Output: S_2 - Next GPS sample in PoA; Sig(S_2, T^-) - Signature of S_2
while *true* **do**
  $S_2 \leftarrow$ `ReadGPS();`
  $z \leftarrow$ `FindNearestZone`(S_2, Z);
  $D_1 \leftarrow$ `Dist`(S_1, z);
  $D_2 \leftarrow$ `Dist`(S_2, z);
  if $S_2.t - S_1.t \leq (D_1 + D_2)/v_{\max} < S_2.t - S_1.t + 2/R$ **then**
    S_2, Sig$(S_2) \leftarrow$ `GetGPSAuth();`
    return S_2, Sig(S_2, T^-);
  else
    sleep$(1/R)$;
  end
end

GPS data was replayed to the GPS Sampler to emulate the real-time GPS samples read from the GPS Driver interface.

We specify two sets of no-fly-zones into the AliDrone client. In the first case, we set a single NFZ with a large radius. This case represents large no-fly areas in the city or nearby critical infrastructures like airports and power plants. In the second case, we set multiple small and dense no-fly-zones along the route of the driving path. This case simulates the scenario where the drone flies through a residential area and it should not fly over any of the neighbors with no-fly-zone.

We compare the adaptive sampling with a baseline approach which we refer as "Fix Rate Sampling". Every time after a GPS data is sampled, the sampling thread will sleep for a period according to the sampling rate. Since the GPS hardware has an independent rate for updating the measurements, the sampler cannot always get the most updated GPS data immediately after it wakes up. Therefore, we let the sampler wait until the first measurement update for each time after it wakes up. As a result, the actual sampling rate is as fast as configured. For example, if the update rate of GPS hardware is 5Hz, five samples are produced in each second at $t = 0.0, 0.2, 0.4, 0.6,$ and 0.8s. If the sampler runs at 3Hz, it wakes up at $t = 0.0, 0.33,$ and 0.67s. Then the time that three samples are taken should be $t = 0.0, 0.4, 0.8$s.

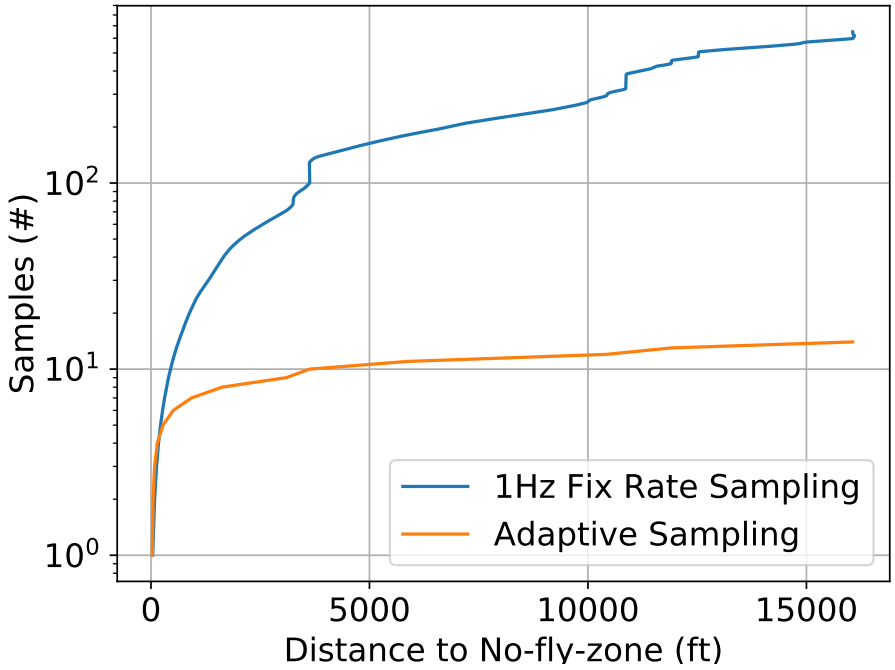


Figure 5.4: In airport scenario, we keep track of the total number of GPS samples, and the distance between the vehicle and the boundary of the NFZ.

Airport Scenario FAA regulations forbid drone operations within 5 miles of any airport. In this scenario, we set an NFZ centered at an airport with a radius of 5 miles. The GPS trace starts about 30 feet outside the boundary of the NFZ. The vehicle drives away from the NFZ for about 3 miles in 12 minutes.

We set the sampling rate as 1Hz and keep track of the total number of GPS samples as well as the distance to the boundary. When the vehicle is close to the boundary, the sampling rates of fix rate sampling and adaptive sampling are similar. As the distance increases, the adaptive sampling requires fewer samples for a sufficient alibi. Comparing to the 649 samples collected by 1Hz fix rate sampling, the adaptive sampling uses only 14 GPS samples.

Residential Scenario The residential areas are comprised of many small but dense NFZs. In this scenario, we drive the vehicle through a local county for about one mile. Figure 5.5 shows the satellite view of the residential area and marks the driving route from location A to B. For purpose of anonymity, the names and labels are removed from the map. We use Google Maps to identify the houses along the driving route and mark each of them as an NFZ. Every NFZ is represented by a circle centers at a house with a radius of 20 feet.

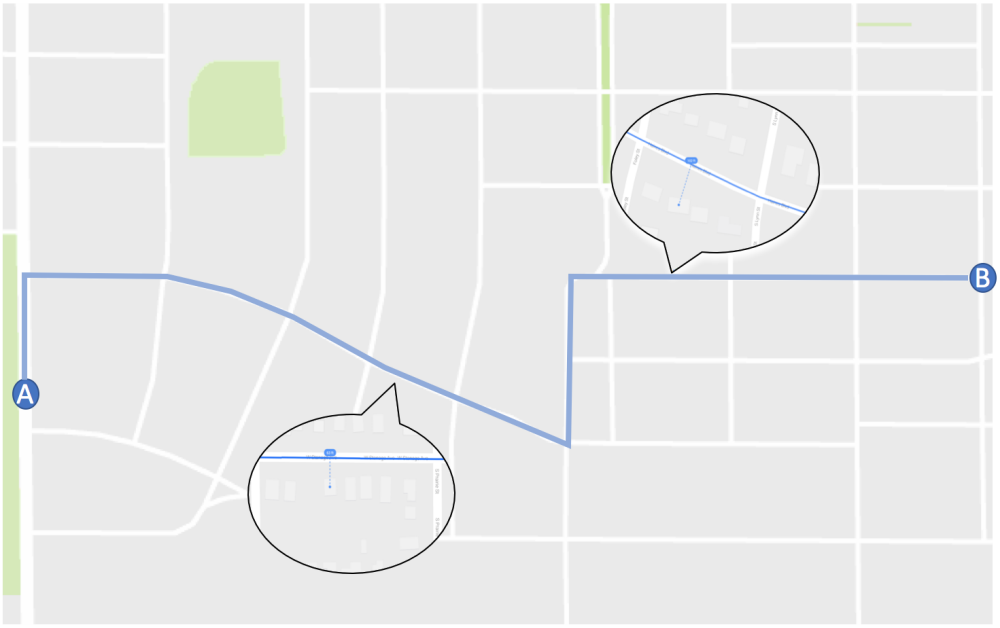


Figure 5.5: Map and driving route of the residential area.

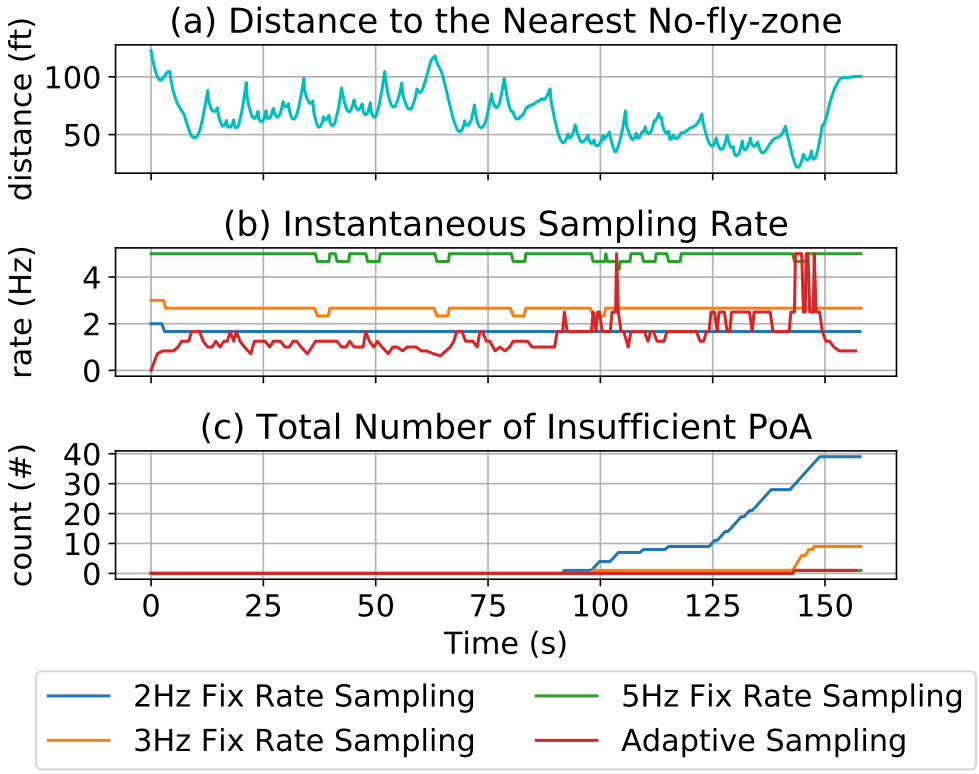


Figure 5.6: We measure three metrics in the residential scenario: (a) distance to the nearest NFZ; (b) instantaneous sampling rate; (c) total number of insufficient Proof-of-Alibi.

In total, 94 NFZs are identified in this area.

We are interested in three metrics in the residential scenario.

Distance to the nearest NFZ: As the vehicle moves, its distances to the NFZs are changing. However, only the nearest NFZ affects the sampling rate because a PoA proving alibi to the nearest NFZ is also sufficient for the other NFZs. The distance of the vehicle to the nearest NFZ is shown in Figure 5.6-(a). Such distance indicates the density of the neighborhood. At the beginning, the distance is primarily within range 50 - 100 ft. When the vehicle enters a more dense area, the distance decreases to 20 - 70 ft. At the closest point, the vehicle is only 21 ft to the boundary of the nearest NFZ.

Instantaneous Sampling Rate: We compare the the instantaneous sampling rate of adaptive sampling to fix rate sampling with 2 Hz, 3 Hz and 5 Hz in Figure 5.6-(b). Note that the sampler may wait for a small period time for the first GPS update, the actual sampling rate in Fix Rate Sampling can be lower than the settings. When the vehicle travels in the less dense area, the Adaptive Sampling uses a sampling rate lower than 2Hz. This saves the total number of GPS samples produced in PoA. As the vehicle enters the dense area, the adaptive algorithm pushes to higher sampling rate to preserve the sufficiency of PoA.

Total Number of Insufficient PoA: If the time between two continuous GPS samples is too long, the trace cannot provide sufficient PoA. For every continuous sample pair (x_i, y_i, t_i) and $(x_{i+1}, y_{i+1}, t_{i+1})$, we count the insufficient PoAs as follows:

$$\text{count +=} \begin{cases} 1 & \text{if } \min_j(d_{i,j} + d_{i+1,j}) \leq v_{\max}(t_{i+1} - t_i), \\ 0 & \text{otherwise,} \end{cases} \tag{5.7}$$

where $d_{i,j}$ is the distance from the location of sample i to NFZ j.

Figure 5.6-(c) demonstrates the total number of insufficient PoAs over time. In the first one and a half minutes, no insufficient PoA is spotted. As the vehicle drives into the dense area, the fix rate sampling with 2Hz and 3Hz are unable to produce sufficient PoA. In total, 39 and 9 insufficient PoAs are counted in 2Hz and 3Hz Fix Rate Sampling.

Adaptive sampling achieves as few insufficient PoAs as fix rate sampling (5Hz). However, an insufficient PoA is identified at a time the vehicle is 25 ft to an NFZ. By further inspection of the GPS trace, we find that the GPS hardware misses an update when insufficient PoA takes place. This means that the maximum sampling rate drops from 5Hz to 2.5Hz at this point.

5.4.2 Benchmarks

In this section, we present the benchmarks of AliDrone by testing the processing and energy overhead in a controlled laboratory environment. The experimental platform of the benchmarks is Raspberry Pi 3 Model B, which has a 1.2 GHz 64-bit quad-core ARMv8 processor and a 1 GB LPDDR2-900 SDRAM memory. The CPU utilization and memory consumption of AliDrone are measured by running the GPS Sampler on a single core under a fixed sampling rate. The power consumption is derived from the power model presented by Kaup et al. [80]:

$$P_{\text{CPU}}(u) = 1.5778\text{W} + 0.181 \cdot u \cdot \text{W} \tag{5.8}$$

where u is the average CPU utilization ranging from 0 to 1.

We first run the GPS Sampler under a fixed sampling rate of 2 Hz, 3 Hz and 5 Hz for 5 minutes. We use `top` command to measure the CPU utilization and memory consumption once per second and take the average over all the measurements. Power consumption is computed by equation (5.8). Two encryption and sign key sizes (1024 and 2048 bits) are tested in the benchmarks. Then, we replay the GPS data collected from the two field studies and run the measurements again using the same settings.

Table 5.3 shows the benchmarks for CPU utilization, power consumption and memory consumption. Since the Raspberry Pi has four cores, the range of CPU utilization measurement is [0, 25%].

Key Size (bits)	Case	CPU (%)	Power (W)
1024	Fixed 2 Hz	2.17 $\pm$0.05	1.5817
	Fixed 3 Hz	3.17 $\pm$0.04	1.5835
	Fixed 5 Hz	5.59 $\pm$0.06	1.5879
	Airport	0.024 $\pm$0.160	1.5778
	Residential	1.567 $\pm$0.827	1.5806
2048	Fixed 2 Hz	10.94 $\pm$0.09	1.5976
	Fixed 3 Hz	16.81 $\pm$0.10	1.6082
	Fixed 5 Hz	-	-
	Airport	0.122 $\pm$0.810	1.5780
	Residential	-	-
Memory		3.27 MB (0.3%)	

Table 5.3: CPU, Power and Memory Benchmarks

The benchmark results show that AliDrone only consumes a small amount of memory of about 0.3%, which suggests that it will not affect other memory intensive tasks.

In terms of CPU utilization, AliDrone can support trustworthy GPS sampling with the maximum rate of 5 Hz using a short sign key (1024 bits). The computation overhead introduced by AliDrone is about 5.6% on average. In the case of large TEE sign key (2048 bits), AliDrone cannot keep up with the maximum sampling rate. This result implies that more efficient signature schemes are required to support higher GPS sampling rate.

The real-world benchmarks demonstrate that the adaptive sampling mechanism can further reduce the processing overhead. Running AliDrone in a dense residential county using a 1024-bit sign key only costs an average of 1.5% CPU cycles. Again the measurement under 2048-bit sign key is not presented because of the large overhead of computing asymmetric signatures.

5.5 SUMMARY

In this section, we demonstrate how the design goals are achieved and summarized on the security features of AliDrone. We claim that the following design goals are achieved:

Completeness As long as the PoAs are sufficient, it implies that the possible traveling range of the drone does not intersect with any NFZ, which means the drone cannot travel into any NFZ at any time.

Low Overhead As shown in evaluation, AliDrone only add to a small amount of power consumption.

Unforgeability The malicious drone provider is unable to forge a PoA because he does not have access to the TEE sign key.

CHAPTER 6: ENFORCING DATA PRIVACY FOR COMMERCIAL DAAS APPLICATIONS

In this chapter, we introduce our solution to enforce data privacy for commercial DaaS applications, *a.k.a* the Secure Homomorphic Encryption (SHE) framework.

6.1 PRELIMINARIES

6.1.1 Table of Notations

Model	S	=	Set of data sites.
	s_i	=	The i-th data site. i from 1 to n.
	(x_i, y_i)	=	The coordinate of task site s_i.
	U	=	Set of drones.
	u_i	=	The i-th drone. i from 1 to m.
	$\mathcal{P}_i$	=	The path assigned to drone u_i.
Keys	(C^+, C^-)	=	Asymmetric sign key pair of the Client.
	(U_i^+, U_i^-)	=	Asymmetric encryption key pair of drone u_i.
	(S_i^+, S_i^-)	=	Asymmetric encryption key pair of data site s_i.
	sk_{ij}	=	Shared secret key established between s_i and u_j.
	$(\mathsf{pk}_C^+, \mathsf{pk}_C^-)$	=	Homomorphic encryption key pair of the Client.
Protocol	$\mathsf{Sig}(M, K)$	=	Sign message M with key K.
	$\mathsf{Enc}(M, K)$	=	Encrypt message M with key K.
	FHE-$\mathsf{Enc}(M, K)$	=	Homomorphically encrypt message M with key K.
	$\mathsf{Dec}(C, K)$	=	Encrypt cipher C with key K.
	$\mathsf{KGen}(x, y)$	=	Generate a shared secret with two randoms x, y.

Table 6.1: Table of Notations Used in This Chapter

6.1.2 System Model

We consider that n data sites $S = \{s_1, s_2, \ldots, s_n\}$ are distributed on a 2D-plane, each site s_i is associated with a coordinate (x_i, y_i). A *Client* contracts with the *Drone Provider* to send m drones $U = \{u_1, u_2, \ldots, u_m\}$ to collect data from these sites. For simplicity, we assume that all the drones take off from the same spot s_0 with coordinate (x_0, y_0). We use a directed graph $G = (V, E)$ to describe the connectivity of these locations. The vertices $V = s_0 \cup S$ represent all data sites (including the take-off spot). The edges $E = \{(s_i, s_j) \mid \forall i \neq j \in \{0, 1, 2, \ldots, n\}\}$ represent the traveling paths taken by a drone from

one site to another. The distance between two vertices (s_i, s_j) is defined as the Euclidean distance $d_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

The drone provider assigns a path $\mathcal{P}_k = \{s_0, s_{k_1}, s_{k_2}, \ldots, s_{k_l}\}$ to each drone u_k to pass through and collect data from a group of data sites. We require that each site is visited and only visited by one drone, i.e.

$$\begin{cases} \mathcal{P}_1 \cup \mathcal{P}_2 \cdots \cup \mathcal{P}_m = V \\ \mathcal{P}_i \cap \mathcal{P}_j = \{s_0\}, \ \forall\, i \neq j \in \{1, 2, \ldots, m\} \end{cases} \tag{6.1}$$

When a drone u_i approaches a site, it follows the SHE data collection protocol to collect the sensor data. After a drone u_i has traversed through all the sites in $\mathcal{P}_i$, it will return the collected data to the drone provider at the take-off spot. The communication protocol and SHE features will be further discussed in section 6.2.

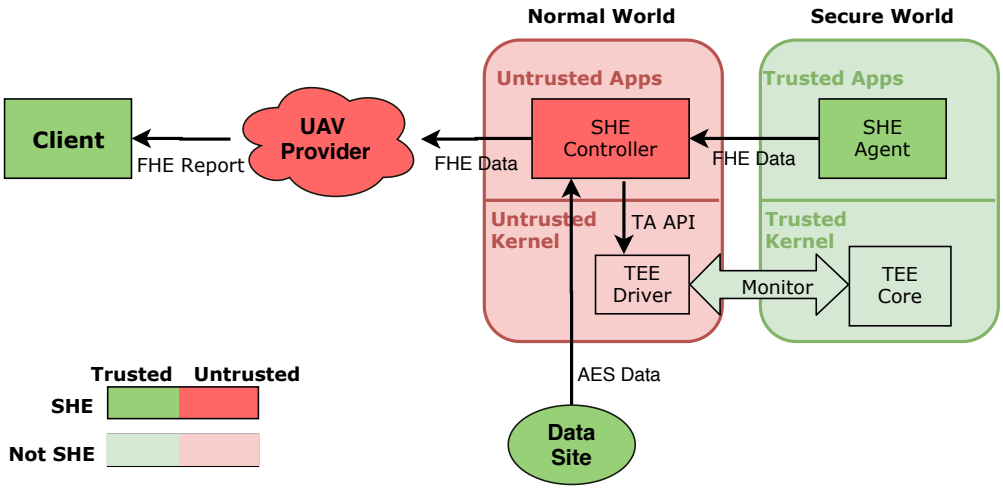


Figure 6.1: SHE system architecture and boundary. The green and red regions illustrate for the trusted and untrusted components. The system boundary covers all dark components.

6.1.3 Threat Model

We consider the adversary as an honest but curious drone provider or pilot that wants to reveal the private data. Such an adversary may be malicious business looking to gain benefits from leaking the sensitive data. An adversary may attempt to obtain plaintext data by, for example, passively reading any accessible hardware buffer or actively eavesdropping with man-in-the-middle (MITM) attack.

We assume that all drone platforms support secure hardware to create a trusted enclave (i.e., ARM Trustzone or Intel SGX). Furthermore, we assume that an asymmetric encryption key pair is generated within the TEE by the hardware manufacturer, and the private key is not known by the drone provider.

6.1.4 SHE Architecture

We present the system boundary and architecture of the Secure Homomorphic Encryption framework from the view of a single drone in Figure 6.1. Looking from the inside of a drone, we divide it into two isolated regions, a trusted "secure world" and an untrusted "normal world".

The *SHE Agent* operates inside of the secure world. It is the only entity that has access to the private TEE key. The SHE Agent is certificated by the Client such that it can read the encrypted messages from the data sites. The major responsibility of the SHE Agent is to *recrypt* the encrypted data coming from the data sites. The recrypt operation involves SHE Agent in transforming the ciphertext encrypted by a symmetric encryption scheme (e.g. AES) into fully homomorphic encryption (FHE) scheme. The format of the collected data is assumed to be numerical numbers. More detail of the recrypt operation is described in Section 6.2.

The *SHE Controller* is a normal world service which is responsible for coordinating and relaying messages between the SHE Agent and the data sites. It interacts with the TEE by calling the Trusted Application APIs provided by the SHE Agent. Note that the drone provider is assumed to have privileged control of the normal world. Thus, a malicious drone provider may be able to monitor the messages sent from or received by the SHE Controller.

When the data collection phase is completed, the drone provider collects the FHE encrypted data from the SHE Controller and executes specific data processing algorithm upon the collected data. Fully homomorphic encryption scheme enables the drone provider to process the ciphertext without directly accessing the data in plaintext. The drone provider thus can present an FHE encrypted result to the Client.

6.1.5 Security Keys

We summarize the notations and usage for the security keys to be used in the SHE protocol in Table 6.2.

Notation	Description	Visibility
C^-	Private Client sign key.	Client
C^+	Public Client verification key.	Public
U_i^-	Private SHE decryption key of u_i.	u_i
U_i^+	Public SHE encryption key of u_i.	Public
S_i^-	Private decryption key of s_i.	s_i
S_i^+	Public encryption key of s_i.	Public
sk_{ij}	Shared secret key of s_i and u_j.	s_i and u_j
pk_C^-	Homomorphic decryption key of Client.	Client
pk_C^+	Homomorphic encryption key of Client.	Public

Table 6.2: Notations of keys and data used by Secure Homomorphic Encryption protocol. Column **Visibility** indicates the entities who have access to the information.

- (C^+, C^-): asymmetric sign key pair of the Client. The sign key is used to certificate the SHE Agent on drones. The data sites can use the public verification key to verify if the drone is allowed to access the data.
- (U_i^+, U_i^-): asymmetric encryption key pair of the SHE agent on drone u_i. They are used to encrypt and decrypt the handshake messages during the communication with the data sites.
- (S_i^+, S_i^-): asymmetric encryption key pair of the data site s_i. They are used to encrypt and decrypt the handshake messages during the communication with the drones.
- sk_{ij}: shared secret key established between s_i and u_j. It is used to encrypt and decrypt data. The candidate encryption method can be any symmetric encryption scheme, e.g. AES.
- $(\mathsf{pk}_C^-, \mathsf{pk}_C^+)$: homomorphic encryption key pair of the Client. They are used to encrypt and decrypt messages in FHE scheme. The private key is only known to the Client so that other parties cannot decrypt the FHE ciphertext.

6.1.6 Design Objectives

The Secure Homomorphic Encryption protocol is designed to protect the privacy of the sensing data while still allow the drone providers to process the collected data. We list our design goals as follows:

Privacy-preserving The data must remain secret to the drone provider, given that a malicious drone provider or pilot may attempt to eavesdrop passively or actively.

Computable The drone provider must be able to run arithmetic computation on the data and present the encrypted computation result to the client.

Low Overhead The protocol should have low computation and communication overhead on the data sites with the assumption that the sensing devices may only have limited computation resource.

6.2 SYSTEM DESIGN

6.2.1 Protocol Workflow

SHE protocol involves three entities: a Client, a drone provider and multiple data sites. Figure 6.2 demonstrates the interactions among these entities. We describe the protocol workflow in this section.

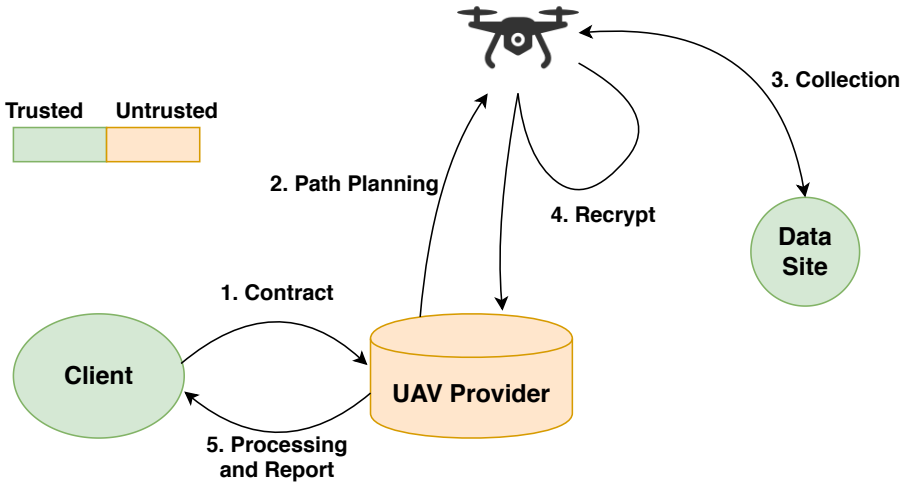


Figure 6.2: An overview of system workflow. The workflow starts where the Client contracts an aerial mission from the drone provider by offering the location of the data sites and certificating the drones (step 1). Then, the drone provider makes path plans for the navigation (step 2). The drones follow SHE handshake and recrypt protocols to collect data from the data sites. (step 3 and 4). When the navigation completes, the drone provider gathers and processes the encrypted data to get the encrypted report back to the client. (step 5).

1. Contract Setup: As the first step, the Client starts by contracting a DaaS task from the drone provider. The Client presents to the drone provider a list of data sites $S = \{s_1, s_2, \ldots, s_n\}$, their coordinates $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, an expiration time of the mission t_e and an algorithm $\mathcal{A}$ to be executed on the data. The drone provider then plans to send a set of drones $\{u_1, u_2, \ldots, u_m\}$ to execute the mission and submits the public encryption keys $\{U_1^+, U_2^+, \ldots, U_m^+\}$ of these drones to the Client for mission certificate.

The Client uses its asymmetric sign key to create a signature on the drones' public keys concatenated by the expiration time of the mission $\mathsf{Sig}(U_i^+ \oplus t_e, C^-)$. This signature will serve as the certificate during the contract.

2. Path Planning: For each drone u_k, the drone provider plans a path for it to travel through a subset of data sites $\mathcal{P}_k = \{s_0, s_{k_1}, s_{k_2}, \ldots, s_{k_l}\}$. Two conditions should be satisfied: (1) the drone must return to the drone provider after it visits all the assigned sites; (2) the drone must be able to complete planned path before the expiration deadline t_e and its battery capacity t_b. The method and algorithm to find the optimal paths is out of the scope of this work.

3. Data Collection: When a drone u_j approaches a data site s_i, it sends a *hello* message from the SHE agent to initiate the one-way handshake. The hello message is defined as the drone's public key concatenated by the certificate obtained from the Client

$$\text{hello} \coloneqq U_j^+ \oplus \mathsf{Sig}(U_j^+ \oplus t_e, C^-) \oplus \mathsf{Enc}(r_j, S_i^+), \tag{6.2}$$

where r_j is a random picked and stored by the SHE Agent.

As the Client's verification key C^+ is public, the data sites can verify the hello message and determine if the data collection request is valid by looking at the drone's public key and the expiration time. If the request is valid, it picks a second random r_i and respond with a *hello-ack* message as follows

$$\text{hello-ack} \coloneqq \mathsf{Enc}(r_i \oplus r_j \oplus \mathsf{pk}_C^+, U_j^+). \tag{6.3}$$

The two randoms will be used to generate a shared secret key $\mathsf{sk}_{ij} = \mathsf{KGen}(r_i, r_j)$. This key will be used to encrypt/decrypt the sensing data. In addition, the hello-ack message binds the shared secret key with the homomorphic encryption key of the Client pk_C^+. This key will be use in the recrypt operation.

4. Recrypt: After the shared secret key is established, the data site s_i can transmit private data to the drone by

$$\text{data} \coloneqq \mathsf{Enc}(\text{raw-data}, \mathsf{sk}_{ij}). \tag{6.4}$$

Note that this message uses the symmetric encryption scheme, it only takes a small amount of time and bandwidth to transmit the encrypted data.

Upon receiving the encrypted data block, the SHE agent can use the shared key to decrypt it, and then encrypt it using the fully homomorphic encryption scheme. We refer to this process as *recrypt* and it can be described as follows:

$$\text{fhe-data} \coloneqq \mathsf{FHE\text{-}Enc}(\mathsf{Dec}(\text{data}, \mathsf{sk}_{ij}), \mathsf{pk}_C^+). \tag{6.5}$$

Finally, the FHE encrypted data will be stored on the drone.

5. Processing and report: When all the drones complete the navigation, the drone provider can collect the FHE encrypted data from all the drones and run the demanded algorithm $\mathcal{A}$ on the ciphertext. Since the data is encrypted by fully homomorphic encryption scheme, the drone provider can perform arbitrary computation on the ciphertext.

6.2.2 Recrypt Protocol v.s. Fully Homomorphic Encryption

The Fully Homomorphic Encryption is very expensive for devices with limited computation resource. Previous studies [51] show that a general computation platform (e.g. PC) can perform FHE encryption up to 100x faster than common IoT devices (e.g. Raspberry Pi). Moreover, the size of FHE ciphertext is huge. Encrypting a 10-bit numerical number with the state-of-the-art FHE libraries such as Helib [48] or SEAL [81] produces a ciphertext with a size of over 60k bytes. This will introduce significant communication overhead. Although bootstrapping operation can be applied to reduce the size of the ciphertext, it usually takes more time than the FHE encryption operation and thus adding heavier computation load to the IoT devices.

Compared to the FHE scheme, the recrypt protocol moves the FHE encryption from the data sites to the drone. The data transmitted is encrypted with symmetric encryption scheme which is very fast and efficient in terms of the size of ciphertext blocks. A standard AES-256-CBC [82] only uses 16-bytes in the ciphertext-size and it only takes less than 1 millisecond on Raspberry Pis to run AES encryption. Furthermore, since the commercial drone platforms on the market have much better computation performance than the sensing devices, it takes less time for the drones to compute FHE encryption than the data sites.

6.3 HARDWARE AND IMPLEMENTATION

6.3.1 Hardware Platform

We choose ARM Trustzone [11] as our secure hardware platform. Although Intel SGX [21] processors provide better performance in general, they do not emulate computation environment of the drone hardwares. TrustZone partitions the software and hardware into two worlds, a.k.a a normal world and a secure world. The hardware logic ensures that the resources in the secure world is inaccessible from the normal world.

Specifically, we implement a proof-of-concept prototype of the SHE framework on Raspberry Pi 3 Model B [83], which has a 1.2GHz 64-bit quad-core ARMv8 CPU that supports ARM TrustZone. Previous effort has shown feasibility of deploying a practical drone controller on Raspberry Pi [84].

6.3.2 Software Interfaces

Now we introduce the implementation of two main components of the SHE.

SHE Agent The SHE agent is implemented as a Trusted Application (TA) in non-privileged mode in the secure world. It uses an asymmetric key pair to decrypt messages from the sensors. Three interfaces are exposed to the SHE Controller: `GenSharedSecret()`, `SetKeys()` and `GetFheData()`.

- `GenSharedSecret()` is used to initiate data collection session. A `session_id` is provided by the SHE Controller. The SHE Agent randomly generates one half of the shared secret r_j and stores it with the session id. The shared secret r_j is returned to the SHE Controller.

- `SetKeys()` is called when a sensor replies a hello-ack message. The full hello-ack message and the session id are feeded to the SHE Agent. The SHE Agent uses its decryption key to get the other half of the shared secret r_i and the homomorphic encryption key pk_C^+ from the hello-ack message. The shared key sk_{ij} is computed with the two halfs of shared secrets. The SHE Agent stores the shared key, the homomorphic encryption key and the session id in the memory.

- `GetFheData()` is called when an AES encrypted data block is received by the SHE Controller. The SHE Controller passes the encrypted data and the session id to the

SHE Agent for recrypt operation. FHE encrypted ciphertext is returned to the SHE Controller.

SHE Controller The SHE Controller is implemented as a daemon service in the user space of the normal world. It is responsible for coordinating messages among SHE Agent, sensors and the drone provider. The main features implemented on the SHE Controllers are as follows:

- `GetCertificate()` is used to obtain certificate from the Client. The SHE Controller presents the public key of its SHE Agent to the Client and get a signed certificate for a mission.
- `StartDcSession()` is used to initiate a data collection session when the drone is getting into the communication range of a sensor. It gets a half secret from the SHE Agent by calling `GenSharedSecret()` and send a hello message to the sensor. Note that although SHE Controller shares the half secret with the SHE Agent, it is not aware of the other half secret because it does not have access to the SHE Agent's decryption key.

  When a hello-ack message is received. The SHE Controller calls `SetKeys()` to establish shared keys between the SHE Agent and the sensors. Upon receiving the follow-up data blocks, it uses `GetFheData()` interface to obtain FHE encrypted data from the SHE Agent. The FHE ciphertext is stored in the memory space in the normal world.

6.4 SYSTEM EVALUATION

In this section, we present the system performance benchmarks of SHE framework in a controlled laboratory environment. The system performance is measured in two phases. In the data collection phase, we focus on the processing and energy overhead of drones to run the encrypt and recrypt operations. In the data processing phase, we design several data processing patterns and measure the computation time and memory consumption of the drone providers.

6.4.1 Data Collection Phase

We measure the CPU utilization and energy consumption during the data collection phase. The experimental platform that emulates a drone is Raspberry Pi 3 Model B, which

has a 1.2 GHz 64-bit quad-core ARMv8 processor and a 1 GB LPDDR2-900 SDRAM memory. We run a "mocked sensor" (MS) on a Unix-based laptop, which has a 64-bit 2.8GHz quad-core Intel Core I7 processor. The MS implements the SHE protocol so that it can respond to the handshake messages and send encrypted numerical data to the drone platform.

We use `top` command to measure the CPU utilization of the SHE Agent and SHE Controller on the drone platform. The CPU utilization is measured once per 100 milliseconds and we take an average on all the measurement readings in a measurement cycle. The same power model for Raspberry Pi is used as in Section 5.4.

Each measurement cycle begins when the drone sends a hello message from the SHE Controller to the MS. Then, the MS replies with the hello-ack message to establish the shared AES key, and uses this key to send k encrypted data blocks to the drone platform. As a result, the SHE Agent will run k recrypt operations to convert the AES encrypted data to FHE encrypted data. Such measurement cycles are executed once per second. If the one measurement cycle cannot be finished before the next cycle starts, we will abort the last cycle and start the next one immediately.

k	CPU (%)	Power (W)
1	9.29 ± 0.54	1.579
5	21.87 ± 1.20	1.617
20	66.17 ± 2.91	1.698
31	99.52 ± 1.03	1.759

Table 6.3: CPU Utilization and Energy Consumption for Data Collection Phase

Table 6.3 shows the experimental result for CPU utilization and power consumption under different number of recrypt operations in one data collection session. The handshake process uses 2048-bit keys and the `RSAES-PKCS1-v1_5` standard to encrypt and decrypt the hello and hello-ack messages. The TFHE library is used in the SHE Agent and the security level of FHE is set to match the 1024-bit asymmetric key.

The experimental result suggests that SHE reaches the processing limitation at around 31 recrypt operations per second with our security settings. It is possible to mitigate this problem an asynchronous implementation.

6.4.2 Data Processing Phase

In the data processing phase, we design several tasks for the drone providers and evaluate the performance of the computation and storage overhead. The performance eval-

uations are run on a desktop computer which has a 64-bit hexa-core Intel I7 processor running at 3.7GHz with 16GB RAM. We choose the following tasks to be executed by the drone provider:

- **Average** of N FHE encrypted data;
- **Dot Product** of an MxN matrix with an Nx1 vector, with each element encrypted by FHE;
- **Linear Classifier** for M input with N attributes by a pre-computed model with K classes. The model and input are both encrypted by FHE.

Task	**Parameters**	**Computation Time**	**Ciphertext-size**	
			Before	After
Average	$N = 10000$	58.4 ms	389.5MB	73.2KB
Dot Product	$M = 100, N = 10$	2067.5 ms	39.3MB	7.47MB
Human-small	$M = 10, N = 37, K = 2$	2127.2 ms	15.9MB	800.6KB
Human-large	$M = 10, N = 561, K = 5$	16517.5 ms	240.5MB	792.3KB

Table 6.4: Performance evaluation for Data Processing Phase. Most of the tasks can be finished on the order of seconds.

These tasks are the common building blocks widely used in many comprehensive data processing algorithms. Specifically for the linear classifier, the models are trained in plaintext using scikit-learn [85], a Python-based machine learning framework. We use the data from two real-world human activity recognition datasets provided by the UCI machine learning repository [86] and train two linear classifier models with different parameters.

We measure the computation time and the ciphertext-size before and after the processing phase and present the evaluation result in Table 6.4. The evaluation result suggests that most of the data processing tasks can be finished on the order of seconds. In addition, Average and Linear Classifier tasks achieve relatively high data compression ratio compared to the Dot Product task.

6.5 SUMMARY

In this section, we demonstrate how the design goals are achieved and summarized on the security features of SHE. We claim that the following design goals are achieved:

Privacy-preserving The drone provider can only access the AES-encrypted data transmitted by the sensors and the FHE-enrypted data generated by TEE. Since he does not have the AES key or the FHE decryption key, he is not able to decrypt the data in plaintext.

Computable The drone provider can run arithmetic operations on the FHE-encrypted data.

Low Overhead Our experiment demonstrates that our proof-of-concept prototype can be successfully executed on a computationally limited device.

CHAPTER 7: ENABLING EFFICIENT AND TRUSTWORTHY CONTRACT MANAGEMENT COMMERCIAL DAAS APPLICATIONS

In this chapter, we introduce the design and implementation of UAVChain to enable efficient and trustworthy contract management for commercial DaaS applications.

7.1 PRELIMINARIES

7.1.1 Table of Notations

Model	C	=	Set of clients.
	c_i	=	The i-th client. i from 1 to M.
	P	=	Set of drone providers.
	p_i	=	The i-th drone provider. i from 1 to N.
	S	=	Set of task sites.
	s_i	=	The i-th task site. i from 1 to n.
	(x_l, y_l)	=	The coordinate of task site s_l.
Protocol	t_b	=	Battery duration of a drone.
	r	=	Amount of reward coins deposit by the client.
	b	=	Amount of bidding coins deposit by the drone provider.
	m	=	Amount of mediate fee deposit by the client.
Formulation	$\mathcal{T}_{i,j}$	=	Time to travel from s_i to s_j.
	T^k	=	Total operation time on the k-th path.
	$x^k_{i,j}$	=	Binary variable. =1 if s_i to s_j is in the k-th path.

Table 7.1: Table of Notations Used in This Chapter

7.1.2 Graph Model

We consider that M clients $C = \{c_1, c_2, \ldots, c_M\}$ and N drone providers $P = \{p_1, p_2, \ldots, p_N\}$ are distributed on a 2D-plane, and denote the coordinates of a drone provider p_i as (x_i^0, y_i^0).

A task from client c_j requires a drone provider to send drones and visit n task sites $S_j = \{s_1^j, s_2^j, \ldots, s_n^j\}$ with no specific order. The coordinates of each task site s_l^j is denoted as (x_l^j, y_l^j). We assume that all the drones owned by the drone provider p_i always take off and return to the same location of p_i during each task.

We use a directed graph $G = (V, E)$ to describe the connectivity of all the aforementioned locations. The vertices $V = P \cup S_1 \cup S_2 \cdots \cup S_M$ represent the locations of all

drone providers and task sites. The edges $E = \{(v_i, v_j) \mid \forall i \neq j, v_i \in V, v_j \in V\}$ represent the traveling path taken from one vertex to the other. The distance between two vertices (v_i, v_j) is defined as the Euclidean distance. When a drone travels from one vertex to another, we always assume the traveling path is a straight line without any obstacle. Plenty of previous research have studied the path planning problem to avoid obstacles [87, 88, 89]. We argue that our approach can adapt to the obstacle scenario by replacing the distances between vertices with the actual traveling distance computed from a obstacle-avoiding algorithm.

7.1.3 Task Workflow

The workflow of a task starts when a client posts a *task description* to an Ethereum-based platform, a.k.a. UAVChain. A task description includes the following information:

- The general description for the purpose of this task;
- The reward for this task in cryptocurrency;
- The deadline that this task needed to be finished before;
- The requirement for the drone specification, e.g. carrying weight, battery hours, etc.
- A list of locations for the task sites that the drones will visit;
- A list of operations that the drones will execute at each task site, e.g. taking a picture;
- The description of the report to be returned to the client;
- A list of proof-of-work to be returned to the client.

The task information will be displayed to the public until it is assigned to a drone provider. The assignment of a task from client c_j to drone provider p_i guarantees that the drone provider p_i has enough drones to traverse all the task sites in S_j. Also, we optimize the assignment algorithm such that the chosen drone provider only uses a minimal number of drones to finish the task before its deadline. We will further discuss the task assignment algorithm in Section 7.3.

Once a task is assigned from c_j to a drone provider p_i, the drone provider p_i will send a couple of drones to traverse all the task sites in S_j. When a drone approaches a task site, it executes the demanded operation according to the task description, and generates

a corresponding *proof-of-work*. The proof-of-work is a verifiable data structure that proves the demanded operation is executed correctly.

UAVChain allows the client to specify four different types of proof-of-work: location, time, image and encrypted data. Depending on task description, the drone provider may be demanded to submit multiple proof-of-works in one task. Taking the parcel delivery as an example, the drone provider should submit a picture that shows the parcel is dropped at the door of the receiver, and a proof of location to prove the correctness of the delivery address. The detailed design and implementation of the proof-of-work is demonstrated in Section 7.2.2.

Upon task completion, the drone provider will submit the report and the proof-of-work to the UAVChain. The client can verify the report and the proof. If the client is satisfied, the amount of the promised cryptocurrency will be transferred automatically from the account of the client to the account of the drone provider. Otherwise, the client can ask a mediator, who is an authorized entity, to step in and resolve the conflict.

7.1.4 Threat Model

We identify three different types of adversaries in this system. First, we consider that a dishonest drone provider can attempt to get the reward from clients without completing the demanded tasks. Such a malicious drone provider may fake the report or proof to bypass the verification step. Second, we consider that a dishonest client can try to avoid paying the promised reward to the drone provider when a task is completed correctly. Last but not least, we consider the adversary as an outsider who impersonates a legitimate participant and aims to gain reward from or take down the UAVChain. A potential attack from such an adversary can be a denial-of-service (DoS) attack, which exploits triggering unnecessary computation flow in the system.

We assume that all drone platforms support secure hardware to create a trusted enclave (i.e., ARM Trustzone or Intel SGX). Furthermore, we assume that an asymmetric encryption key pair is generated within the TEE by the hardware manufacturer. The drone provider has the public key but does not have access to the private key.

7.1.5 Design Objectives

The UAVChain is designed to assign DaaS tasks in an efficient and trustworthy manner. Our solution leverages the trusted execution environment (TEE) on the drones to

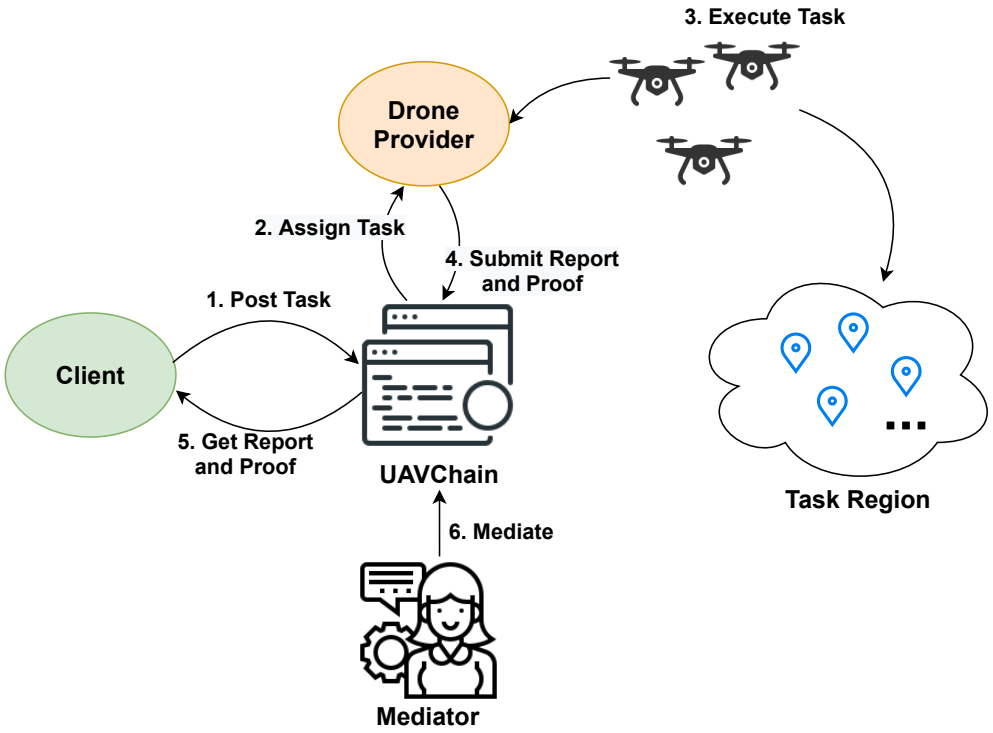


Figure 7.1: An overview of system workflow. The workflow starts where a client post a task to the UAVChain using the smart contract API (step 1). Then, the drone providers bid for the task and one of them wins the task and get the assignment (step 2). The drone provider sends a couple of drones to execute the task (step 3). When the task is finished, the drone provider gathers the report and proof from the drones and submit to the UAVChain (step 4). The client can get the report and verify the proof (step 5). If the client is not satisfied with the result, a Mediator will step in to mediate between the client and drone provider to resolve the conflict (step 6).

provide trustworthy proof-of-work, and utilizes smart contracts to manage the contract information on a distributed ledger. The main design goals of UAVChain are listed below:

Efficiency UAVChain must be efficient in terms of system performance. In addition, it should use minimal resource to complete the tasks.

Verifiability The task result returned to the clients must be trustworthy and verifiable so that the client can distinguish if the drone provider is a potential adversary.

Correctness UAVChain must resolve the reward distribution correctly regardless of the completion of the tasks. Given the existence of adversaries, UAVChain should be able to detect and punish malicious users as well as to protect the benefits of honest participants.

7.2 SYSTEM DESIGN

7.2.1 System Workflow

We describe the workflow of UAVChain in this section using an example of one client and one drone provider. Figure 7.1 demonstrates the interactions among the participants in the protocol.

```
struct TaskDescription {
  string public info;
  // The general description for the task

  uint256 public reward = r;
  // The amount of reward paid to the drone provider

  uint256 public expiration;
  // Deadline of this task

  DroneSpec public spec;
  // The required specification for the drones

  Location[] public locations;
  // The coordinates of task sites, expressed as (lat, lon) pairs

  string public operation;
  // Operation taken by drones at each task site

  string public report;
  // Description of report returned to the client

  enum PoWType { LOCATION, TIME, IMAGE, ENC_DATA }
  PoWType[] public types;
  // Types of proof-of-work
}
```

Listing 7.1: TaskDescription Data Structure

1. Post Task: In the first step, the client starts by posting a task description on the UAVChain. The data structure of a task description is shown in Listing 7.1. It includes a general description for the task, an amount of the promised reward, a task deadline, a specification of the drone requirement, a list of the locations for the task sites, the operation of the drones, a description of the report, and the types of proof-of-work. When the client posts a task, she is required to deposit r coins in the UAVChain, which is the

promised reward stated in the task description. Once the transaction of task posting is processed, the information of this task becomes public to all the potential drone provider for a t_b hours. The client will not get back the coins unless (1) the task is not assigned to any drone provider within time t_b, or (2) the assigned drone provider fails to complete the task.

2. Assign Task: Any drone provider who is willing to take a task can bid for it by depositing a number of coins b. Since we do not consider the game theory based strategies, we simply let $b = r$. While bidding for a task, the drone provider is required to provide the following information:

- The number of drones he can offer;
- The drone specification;
- The location of the drone provider.

After t_b hours since the task is posted, a scheduled transaction will be triggered to run the task assignment algorithm. The algorithm aims to find the drone provider who can complete the task with minimal number of drones. In addition, the task assignment algorithm will give a suggested route to the drone provider.

3. Execute Task: Upon getting the assignment, the drone provider sends a swarm of drones as instructed by UAVChain to traverse all the task site. When a drone approaches a task site, it executes the demanded operation as described in the task description. The report and proof are obtained at the same time by in the trusted execution environment (TEE) on the drone.

4/5. Submit/Get Report and Proof: When all the drones return to the drone provider, the drone provider gathers all the report and proof from the drones and submit them to the UAVChain. In addition, the public keys of the drone TEE are also uploaded so that the client can verify the cryptographic signatures generated by the drone TEE. The report and proof are encrypted and presented through a private url. Only the client can view these contents with her credential. If the client is satisfied with the report and proof, the promised r coins will be transferred to the drone provider.

6. (Optional) Mediate: If the client is not satisfied with the report and proof presented by the drone provider, she can ask a Mediator to step in. We assume that the mediator is an authorized individual or the quorum of a group of judges such that it does not collude with either client or drone provider. To initiate the mediate step, the client is required to deposit m coins as the mediate fee. If the mediator decides that the client wins, the client can get all her deposit $r + m$ coins back, and the drone provider will be responsible to pay

the mediate fee m. Otherwise if the drone provider wins, the client will lose the all of her deposit. The drone provider will get r coins as the reward and the mediator will get m coins.

7.2.2 Proof-of-work Design

The construction of proof-of-work follows the similar idea of AliDrone and SHE. UAVChain supports four types of proof-of-work: location, time, image and encrypted data. We briefly describe the data format and the use case of each type of the proof.

Location A proof of location is constructed as a latitude-longitude pair of GPS coordinates and a TEE signature on the coordinates. It can be used to show that the drone has visited the authenticated location. When combined with other proofs, it can also prove that the drone has executed certain operations at this location.

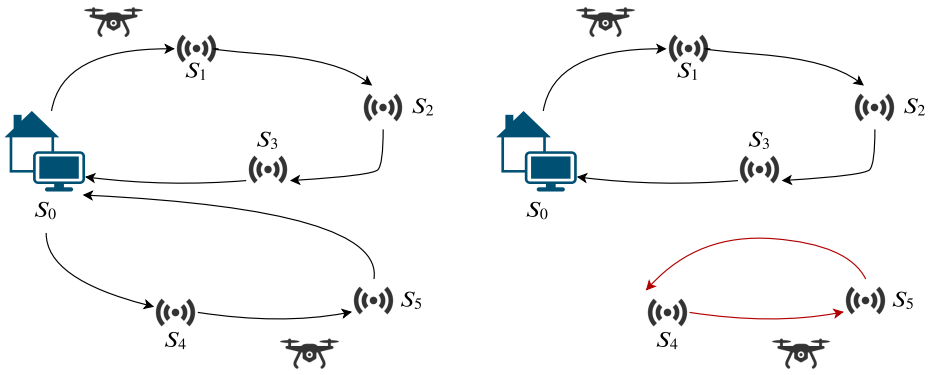
Time A proof of time is constructed as a timestamp and a TEE signature on the timestamp. It can be used to show that the drone is operated at the authenticated time.

Image A proof of image is constructed as an image and a TEE signature on the hash of the image file. It can be used to authenticate that the image is taken by this drone.

Encrypted Data A proof of data is constructed as the ciphertext of the data and a TEE signature on the decrypted data. It can be used to prove the data source when the task requires the drones to collect or transmit data with other devices.

7.3 TASK ASSIGNMENT PROBLEM

When multiple drone providers bid for the same task, we aim to choose the drone provider who can complete the task with the minimal number of drones (Goal Efficiency). A commercial drone can only travel for tens of miles due to the limitation of the battery hour. To simply the problem, we assume that all the drones have identical battery hour and traveling speed.



(a) A feasible path planning (b) An infeasible path planning

Figure 7.2: Examples of feasible and infeasible path planning. The dummy variables are introduced to avoid producing an infeasible solution.

7.3.1 Path Planning Formulation

To find the drone provider best fits the efficiency goal, we formulate the path planning problem for one task and one drone provider using integer programming. We consider that all the drones will departure from the location of the drone provider and traverse n task sites over the task region. Denote s_0 be the departure location and the other n task sites being $\{s_1, s_2, \ldots, s_n\}$. The drone provider needs multiple drones and assign each drone to visit a subset of the task sites. As shown in Figure 7.2(a), the drone provider need to send two drones to visit all the five task sites.

The objective of our path planning problem is to send the least possible number of drones so that all the task sites are visited before any drone runs out of battery t_b. Let $x_{i,j}^k$ be a binary variable representing whether the k-th planned path travels from task sites s_i to s_j. Thus,

$$x_{i,j}^k = \begin{cases} 1 & \text{if } s_i \to s_j \text{ is in the } k\text{-th planned path;} \\ 0 & \text{otherwise.} \end{cases} \tag{7.1}$$

Specifically, we let $x_{i,i}^k = 0,\ \forall i, k$ to avoid self-loop. For example, in Figure 7.2(a), there are two trajectories planned, which are $s_0 \to s_1 \to s_2 \to s_3 \to s_0$ and $s_0 \to s_4 \to s_5 \to s_0$. Therefore we have $x_{0,1}^1 = x_{1,2}^1 = x_{2,3}^1 = x_{3,0}^1 = 1$ for the first trajectory and $x_{0,4}^2 = x_{4,5}^2 = x_{5,0}^2 = 1$ for the second trajectory. All other $x_{i,j}^k$s are equal to zero. We further define $x_{i,j} = \sum_{k=1}^{n} x_{i,j}^k$ being the binary variable representing whether there exists a planned path containing the trajectory from s_i to s_j.

Besides $x_{i,j}^k$ and $x_{i,j}$, we also introduce a set of dummy variables $0 \leq z_i^k \leq n \in \mathbb{Z}$, where

$1 \leq k, i, \leq n$. z_i^ks are used to enforce the topology property that any planned path must go through the departure location s_0. As shown in figure 7.2(b), this path planning is infeasible because there is a subloop $s_4 \rightarrow s_5 \rightarrow s_4$ which does not go through s_0. The z_i^ks are introduced and further used in equation 7.6 to avoid getting such an infeasible solution.

We assume all the drones have the same battery hour t_b and flying speed. With the location information of $s_0, s_1, \ldots, s_n$, it is easy to obtain the single trip time from any task site to anther. For simplicity, we denote $\mathcal{T}_{i,j}, \forall i \neq j \in \{0, 1, \ldots, n\}$ being the time for a drone to travel from s_i to s_j. Furthermore, we require every drone to come back to the departure location at the end of the navigation. Besides traveling time, every drone needs to hover above a task site for a small period of time e to perform the demanded operation. Therefore, one intuitive constraint is that the total operation time on every planned path, denoted as T^k, can not exceed the battery hour of any drone, where T^k is defined as

$$T^k = \sum_{i=0}^{n} \sum_{j \neq i} \mathcal{T}_{i,j} x_{i,j}^k + e(\sum_{i=0}^{n} \sum_{j \neq i} x_{i,j}^k - 1) \;\; \forall k \geq 1. \tag{7.2}$$

We thus show the formal formulation as follows.

$$\min \sum_{j=1}^{n} x_{0,j} \tag{7.3}$$

$$\text{s.t.} \sum_{i=0}^{n} x_{i,j} = \sum_{l=0}^{n} x_{j,l} = 1 \qquad \forall\, j \neq 0 \tag{7.4}$$

$$\sum_{i=0}^{n} x_{i,j}^k = \sum_{l=0}^{n} x_{j,l}^k \qquad \forall\, j, \forall\, k \tag{7.5}$$

$$z_i^k - z_j^k + (n+1) x_{i,j}^k \leq n \qquad \forall\, i, j \neq 0, \forall\, k \tag{7.6}$$

$$T^k \leq t_b \qquad \forall\, k \tag{7.7}$$

The intuition of these constraints can be interpreted as follows:

(7.4) Every task site will be included exactly by one path. Therefore, every task site will be visited by a drone and no redundant drone will visit the same task site.

(7.5) For the k-th path $\mathcal{P}_k$, the in degree and out degree on $\mathcal{P}_k$ will be the same for any node. That is, if a task site s_i is on $\mathcal{P}_k$, then its in and out degree on this path are both equal to one. On the other hand, if s_j is not on $\mathcal{P}_k$, its in and out degree on

this path are both equal to zero.

(7.6) The dummy variable z_i^ks are formulated to enforce that the departure site is included in every planned path, which means that all the drones will eventually come back to the drone provider.

(7.7) The time constraint which enforce the total operation time (traveling time plus the execution time for each task sites) cannot exceed the battery hour of a drone.

7.3.2 Heuristic Algorithm

The formulated problem is NP-hard. We design a heuristic algorithm to approximate the path planning problem.

In the worst case, a trivial solution to the path planning problem is just to send n drones, each visiting one task site correspondingly and coming back to the departure spot. In this solution, the worst total traveling time is

$$T_{\text{worst}} = 2\sum_{i=1}^{n} \mathcal{T}_{0,i}. \tag{7.8}$$

In order to minimize the number of drones in the assignment, we introduce two heuristics:

1. Each drone should make maximal contribution to minimize the worst total time.

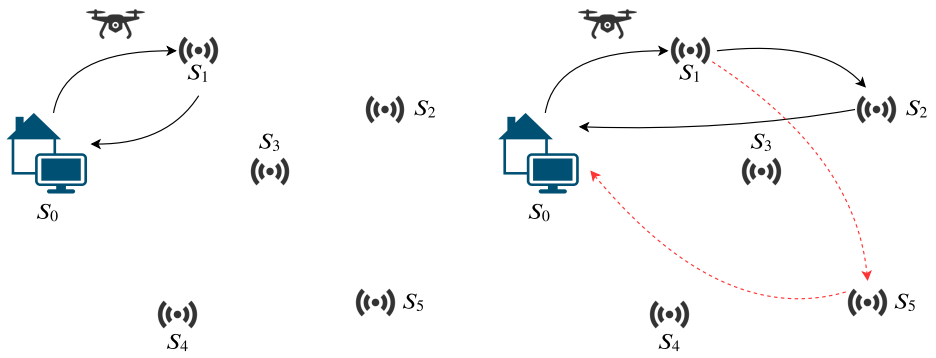
2. The task sites close to each other should be assigned to the same drone.

Suppose we already assigned a path $\mathcal{P}_k = \{s_0, s_{k_1}, ..., s_{k_l}\}$ to a drone. The worst total time for the rest of the task sites is reduced by

$$\Delta T_{\text{worst}} = 2\sum_{i=1}^{n} \mathcal{T}_{0,i},\ \forall\ s_i \in \mathcal{P}_k. \tag{7.9}$$

When we want to assign a new site s_l to the same path, the worst total time can be further reduced by $\Delta T_{\text{worst}} = 2\mathcal{T}_{0,l}$. Heuristic 1 attempts to maximize ΔT_{worst} in the assignment. As a result, the farthest sites will be first assigned to this path.

However, the heuristic 1 alone may assign a very far task site to the path. An example is shown in Figure 7.3. Suppose we already assigned a path $\mathcal{P}_1 = \{s_0, s_1\}$ and we want to assign the next site to this path. According to heuristic 1, the farthest site is s_5. However, a



(a) The current path assignment $\mathcal{P}_1 = \{s_0, s_1\}$.

(b) Assign the next site to $\mathcal{P}_1$ using the heuristics.

Figure 7.3: Examples of path assignment using the heuristics. The heuristic 1 alone may assign a very far site to the path (s_5 in (b)). Combined with heuristic 2, s_2 can be selected as a better assignment.

better assignment should be s_2 in this scenario because s_2 is the second-farthest site from s_0 and also s_2 is close to the current path.

To approximate heuristic 2, we use the idea of *incremental path cost* to compute the extra cost introduced by assigning a new site s_j to a current path $\mathcal{P}_k$. Denote the current path cost of $\mathcal{P}_k = \{s_0, s_{k_1}, ..., s_{k_l}\}$ as

$$T_{\mathcal{P}_k} = \sum_{i=1}^{l-1} \mathcal{T}_{k_i,k_{i+1}} + \mathcal{T}_{0,k_1} + \mathcal{T}_{l,0}. \tag{7.10}$$

Since the new site s_j can be inserted between any two neighbor spots, i.e. changing the path from $s_{k_i} \to s_{k_{i+1}}$ to $s_{k_i} \to s_j \to s_{k_{i+1}}$, we define the incremental path cost as the minimal incremental cost to insert the new site at arbitrary location

$$\Delta T(\mathcal{P}_k, s_j) = \min \begin{pmatrix} T_{\mathcal{P}_k} - \mathcal{T}_{0,k_1} + \mathcal{T}_{0,j} + \mathcal{T}_{j,k_1}, \\ T_{\mathcal{P}_k} - \mathcal{T}_{k_l,0} + \mathcal{T}_{k_l,j} + \mathcal{T}_{j,0}, \\ \min_{i=1}^{l-1}(T_{\mathcal{P}_k} - \mathcal{T}_{k_i,k_{i+1}} + \mathcal{T}_{k_i,j} + \mathcal{T}_{j,k_{i+1}}) \end{pmatrix} \tag{7.11}$$

Selecting a task site with the minimal incremental path cost results in choosing a site that is close to the current path $\mathcal{P}_k$.

We list the heuristic algorithm in Algorithm 7.1. The algorithm `AssignPath()` started with a set of all the task sites and an empty assignment plan. In each iteration, it creates a new path and assigning sites one by one to the path. The criteria of selecting the new

Algorithm 7.1: The algorithm `AssignPath()` searches for a feasible assignment with minimal number of drones.

`AssignPath` ($\mathcal{T},\ S,\ e,\ t_b$):
Input : $\mathcal{T}$ - matrix for traveling costs; S - task sites; e - cost for operation; t_b - battery hour.
Output: $\mathcal{P}$ - an assignment for m drones to traverse $|S|$ spots.

$m \leftarrow 0$; ▷ The number of drones.
$\mathcal{P} \leftarrow \{\}$;
while *S is not empty* **do**
  $m \leftarrow m + 1$;
  $\mathcal{P}_m \leftarrow \{s_0\}$;
  $C_m \leftarrow 0$;
  while *true* **do**
    $i \leftarrow \arg\min_j(\Delta T(\mathcal{P}_m, s_j) - 2\mathcal{T}_{0,j})$; ▷ Minimizing heuristics
    if $C_m + \Delta T(\mathcal{P}_m, s_i) + e > t_b$ **then**
      break; ▷ A new site cannot be assigned to this path.
    end
    $C_m \leftarrow C_m + \Delta T(\mathcal{P}_m, s_i) + e$;
    $\mathcal{P}_m$.insert(s_i); ▷ Minimizing incremental path cost.
    $S \leftarrow S \backslash s_i$;
  end
  $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}_m$;
end

sites is to optimize the two heuristics. As we want to maximize ΔT_{worst} (heuristic 1) and minimize $\Delta T(\mathcal{P}_k, s_j)$ (heuristic 2), we combine them as

$$H \coloneqq \Delta T(\mathcal{P}_k, s_j) - \Delta T_{\text{worst}}, \tag{7.12}$$

and minimizes H in each assignment. The algorithm repeats assigning a task site to the path until the cost of this path exceeds the battery hour t_b. If there are sites not assigned to any path, a new path will be created, which means the drone provider needs to commit one more drone to the task. When the algorithm terminates, all the task sites are assigned to the paths. As a result, m is the minimal number of drones needed to complete the task, and $\mathcal{P}$ is a feasible assignment of these drones. The complexity of this algorithm is $O(n^2)$.

With Algorithm 7.1, we can compute the most suitable drone provider for a task in the smart contract code. Since it is hard to estimate the cost for operation (e), we can set $e = 0$. When a potential drone provider bids for a task, the contract will have its location and set it as s_0. Combining it with the locations of the task sites, we can get the set of all

the sites S and compute the matrix for the traveling costs $\mathcal{T}$. Then, the contract runs the AssignPath algorithm on these parameters to get the minimal possible number of drones and a feasible path assignment.

7.4 SYSTEM EVALUATION

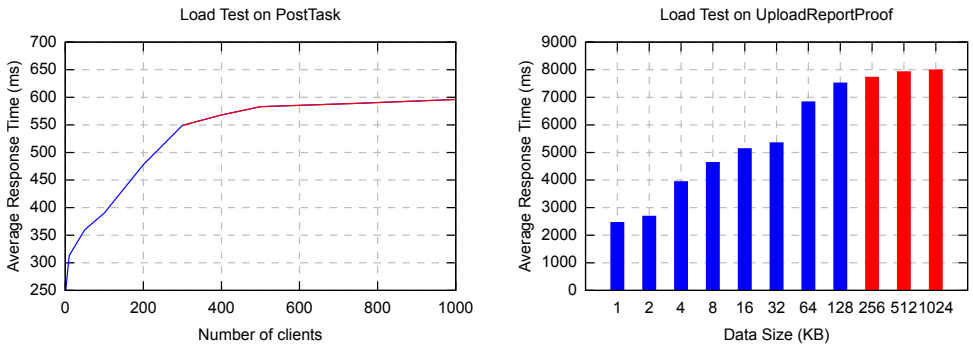
7.4.1 Load Test on UAVChain Prototype

In this section, we present the system performance benchmarks of UAVChain in a controlled laboratory environment. We implement a proof-of-concept prototype for UAVChain in Solidity and enable the interfaces for the participants to post a task, bid for a task, get task assignment and submit report and proof. We deploy the blockchain miners on a local cluster of four VMs. All the VMs are configured with 8 virtual CPUs and 16 GB of RAM.

We also deploy a web server on another local PC with a 3.6-4.9 GHz 8-core Intel i7-9700K processor. The server runs Ubuntu 16.04 64-bit and Apache Web Server 2.4.43. The web server accepts http requests from the clients and drone providers and runs smart contract code in the backend. It provides four interfaces to the users: PostTask, BidTask, UploadReportProof, GetReportProof.

We use Apache JMeter 5.4.1 to load test the prototype and measure the average response time of the web server. In the first scenario, we simulate a load where PostTask is requested by N clients within 5 minutes, with N varied from 1 to 1000. In the second scenario, we simulate that 100 drone providers upload the report and proof through the UploadReportProof interface. We test on different size of the payload from 1KB to 1MB. From our experience, this range covers the sizes of the common types of report and proof. The small proofs (e.g. location) are usually less than 1 KB, and the large proofs (e.g. image) are around 1MB.

The experiment results of the load tests are shown in Figure 7.4. The blue lines and charts show the region where our system handles the requests with negligible number of failures. As the number of client grows over 300 and the size of uploaded payload goes over 128 KB, we observe a significant increase in the failures. This indicates that the system is overwhelmed.



(a) Load Test on PostTask (b) Load Test on UploadReportProof

Figure 7.4: Load test on the proof-of-concept prototype of UAVChain. The blue lines and charts demonstrate that the system can handle all the requests. The red lines and charts show that the system is overwhelmed.

7.4.2 Simulation on Path Planning

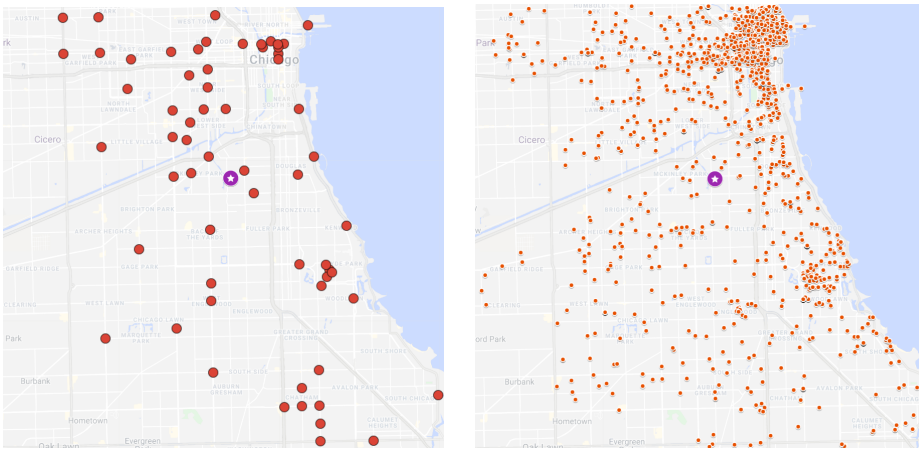
In this section, we present the simulation results of our path planning algorithm on two real-world datasets. As shown in Figure 7.5, the sparse map is retrieved from Chicago Array of Things project [90], and the dense map is retrieved from Chicago Energy Benchmarking project [91]. We randomly select from the locations of the sensors as the task sites. In the sparse map, there are total 63 task sites spanning over the area of Chicago city, which is roughly 300 mile^2. There are total 1600 task sites in the dense map as shown in figure 7.5(b) over the same area. In both two maps, task sites are presented with red dots. We mannually select the location of drone provider on the map, represented as a purple pentagram. All the drones will departure from and return to this location for the task.

In our simulation, we set the battery hour of the drones to 30 mins and the flying speed to 40 mph. We further set the task operation time e to be 500 ms, 800 ms and 1000 ms corresponding to different tasks. We compare the performance of our path planning algorithm with the state of art optimization solver [92] on the sparse map. We do not apply the solver on the dense map because the number of task sites is too large to be solved with the solver in reasonable time. For the dense map, we randomly sample different sized subsets from all the 1600 task sites and apply our path planning algorithm on them. We thus present how the algorithm performance changes corresponding to different number of task sites. The detailed simulation settings are presented in Table 7.2.

For the sparse map, both our algorithm and the solver obtain the minimum number of drones to be 5, which is the optimal solution. However, our path planning algorithm is

	Compare with solver	Number of task sites	Execution time (ms)
Sparse map	✓	63	500
Dense map	×	100, . . . , 1500	500, 800, 1000

Table 7.2: Simulation settings



(a) Sparse Map (b) Dense Map

Figure 7.5: Two Real World task sites Map

significantly faster than the solver. The solver obtains the solution in $163,534$s, while our algorithm only takes 0.043s.

For the dense map, we present the number of drones calculated by our algorithm corresponding to different number of task sites in Figure 7.6. It is easy to observe that even the number of task sites increase dramatically, the number of drones does not increase too much. This is because most of the task sites are assigned to the same path. Another interesting observation is that, when the task sites become denser, an increase of execution time is more likely to result in more drones needed. The reason is that when the task sites are deployed very dense, the total execution time becomes a more important source of drone operation time. While in the case when the task sites are sparse, the drone traveling time is the dominant source of the total navigation time.

7.5 SUMMARY

In this section, we demonstrate how the design goals are achieved and summarized on the security features of UAVChain. We claim that the following design goals are achieved:

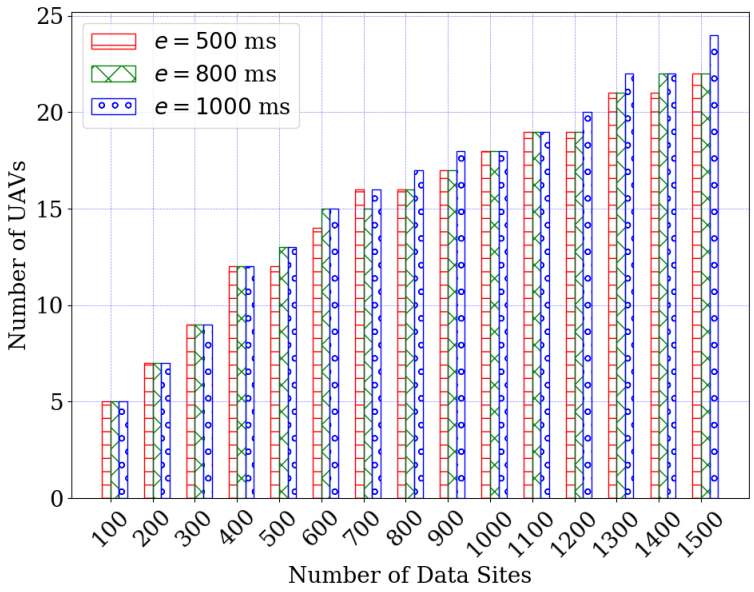


Figure 7.6: Path planning algorithm results with different number of task sites

Efficiency Our system benchmark demonstrate that UAVChain can survive large concurrent loads. Also, due to our task assignment algorithm, the tasks are always executed by the drone provider who can finish the task with minimal number of drones.

Verifiability The PoW enables client to verify the result with the corresponding proofs.

Correctness The mediate protocol ensures that the dishonest participant is punished. The final distribution of the coins is guaranteed to be correct thanks to the smart contract.

CHAPTER 8: CONCLUSION AND FUTURE WORK

8.1 SUMMARY

In this thesis, I present to enforce the privacy compliance of DaaS applications through a combination of hardware-assisted security extension and various security tools. By introducing the hardware-assisted security extension into the DaaS applications, we have an external source of trust so that it enables drones to generate trustworthy proofs and data.

Following this principle, I designed three systems, i.e. Alidrone, SHE and UAVChain, to tackle the location privacy, data privacy and management problems of DaaS applications correspondingly. In each problem, I implement a proof-of-concept prototype and evaluate the system performance in a controlled laboratory environment. The benchmark results show that although the system performance is bounded by certain security parameters, the presented approaches can meet the requirement in the normal use cases.

8.2 LESSONS LEARNED

Throughout the design and experiments in this thesis, I learned that the solution derived from HAPE strategy often encounters a tradeoff between system performance and certain security parameters. For example, the maximum number of trusted GPS samples taken per second is constrained by the length of the encryption and sign key. When the key is longer than 2048 bits, the drone CPU cannot keep up with the GPS sampling rate. Similar results can also be observed in the system benchmark of Recrypt operation in Chapter 6.4.

Although these designs have their performance bottlenecks, there are many potential solutions to improve the system performance. One direct solution is to find the minimum security parameters that are sufficient in different use cases. In addition, the hardware improvement of the future SoCs and hardware-assisted security extensions will help to raise the performance bottleneck. Some potential strategies are also mentioned in the following section.

8.3 FUTURE WORK

8.3.1 Optimization on System Performance

Inspired by the fact that the performance of hardware-assisted approaches are bounded by certain security parameters, one potential direction of the future work may consider optimizing the system performance in the following ways.

Using Symmetric Cryptography Asymmetric encryption is used in both Alidrone and SHE, which consumes a non-negligible amount of time if the number of encryption operations increase. One solution is to use ephemeral symmetric keys between the drone TEE and other entities.

Batching Sign & Encrypt Operations In the design of all three systems, we do not consider batching the operations like sign and encryption. Actually, there are plenty of ways that we can batching technique to optimize system performance. For example in SHE, when TEE receives data from the sensors, it can store it temporarily without running homomorphic encryption right away. Later when it collects more data, it can run aggregation first and then encrypt the result using homomorphic encryption. In this scenario, TEE can save a huge amount of time on the expensive homomorphic encryption.

8.3.2 Large Scale Tasks

One limitation of the UAVChain is that if a task requires a huge number of drones, UAVChain may not be able to find a single drone provider with enough resource to complete the task. In this situation, a client may consider splitting the task into small non-relevant sub-tasks and publish them as standalone tasks. These tasks will be bid separately and assigned to multiple drone providers.

The drawback of the aforementioned solution is that the client has to track multiple contracts with different drone providers. We can solve this problem by allowing drone providers to offload part of the tasks to other drone providers. For example, if a drone provider A is assigned a task which requires ten drones, it can complete part of the task using its five drones, and then publish a task including the other five drones to the UAVChain. Another drone provider B will execute this new task and return the reports and proofs to A. Finally, A will submit all the collected results to the original task. In this solution, the client is only contracted with drone provider A.

8.3.3 3D Physical Model and Obstacles

For model simplicity, we assume the drone can travel from any site to another without any obstacle in our path planning module. The traveling distance between two sites is calculated as the 2D Euclidean distance. However, the drones may encounter obstacles, such as trees, buildings and No-Fly Zones, and thus change their flying height from time to time in real world.

Some possible solutions to deal with the problems in 3D physical world include (1) More precise map of the task area including obstacles, (2) Deploying traditional robotic obstacle avoidance algorithm, i.e. visibility graph algorithm [93], to adjust trajectory when drones are close to obstacles.

REFERENCES

[1] "Amazon air prime," Oct. 2019, https://goo.gl/3NHNre.

[2] "How drones are being used in 2016," Oct. 2019, https://goo.gl/ZbQirw.

[3] P. J. Hiltner, "Drones are coming: Use of unmanned aerial vehicles for police surveillance and its fourth amendment implications, the," *Wake Forest JL & Pol'y*, vol. 3, p. 397, 2013.

[4] T. Krajník, V. Vonásek, D. Fišer, and J. Faigl, "Ar-drone as a platform for robotic research and education," in *International Conference on Research and Education in Robotics*. Springer, 2011, pp. 172–186.

[5] R. Calo, "The drone as privacy catalyst," *Stanford Law Review Online*, vol. 64, pp. 29–33, 2011.

[6] M. Bonetto, P. Korshunov, G. Ramponi, and T. Ebrahimi, "Privacy in mini-drone based video surveillance," in *2015 11th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*, vol. 4. IEEE, 2015, pp. 1–6.

[7] R. Luppicini and A. So, "A technoethical review of commercial drone use in the context of governance, ethics, and privacy," *Technology in Society*, vol. 46, pp. 109–119, 2016.

[8] R. L. Finn and D. Wright, "Privacy, data protection and ethics for civil drone practice: A survey of industry, regulators and civil society organisations," *Computer Law & Security Review*, vol. 32, no. 4, pp. 577–586, 2016.

[9] "FAA summary of small unmanned aircraft rule (part 107)," Oct. 2019, https://goo.gl/JbpgST.

[10] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for cpu based attestation and sealing," in *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, vol. 13, 2013.

[11] "Arm tee reference documentation," Oct. 2019, https://goo.gl/TGq7Kg.

[12] Lenaro, "Op-tee," Oct. 2016, https://goo.gl/53ZVmy.

[13] J. Winter, "Experimenting with arm trustzone–or: How i met friendly piece of trusted hardware," in *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, 2012, pp. 1161–1166.

[14] M. Pirker and D. Slamanig, "A framework for privacy-preserving mobile payment on security enhanced arm trustzone platforms," in *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, 2012, pp. 1155–1160.

[15] Y. Zhang, S. Zhao, Y. Qin, B. Yang, and D. Feng, "Trusttokenf: A generic security framework for mobile two-factor authentication using trustzone," in *Trustcom/BigDataSE/ISPA, 2015 IEEE*, vol. 1. IEEE, 2015, pp. 41–48.

[16] R. Liu and M. Srivastava, "Protc: Protecting drone's peripherals through arm trustzone," in *Proceedings of the 3rd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications*. ACM, 2017, pp. 1–6.

[17] M. Hasan and S. Mohan, "Protecting actuators in safety-critical iot systems from control spoofing attacks," in *Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things*, 2019, pp. 8–14.

[18] "Faa become a pilot," Dec. 2019, https://www.faa.gov/pilots/become/.

[19] "Drone up," Dec. 2019, https://www.droneup.com/.

[20] "Precision hawk oil and gas inspection," Dec. 2019, https://www.precisionhawk.com/oilandgas.

[21] Intel, "Intel software guard extensions (sgx) sdk," Oct. 2019, https://goo.gl/vp3Xm3.

[22] "Op-tee trusted application," May 2017, https://goo.gl/72ebW1.

[23] C. Gentry, S. Halevi, and N. P. Smart, "Homomorphic evaluation of the aes circuit," in *Advances in Cryptology–CRYPTO 2012*. Springer, 2012, pp. 850–867.

[24] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data." in *NDSS*, vol. 4324, 2015, p. 4325.

[25] H. Takabi, E. Hesamifard, and M. Ghasemi, "Privacy preserving multi-party machine learning with homomorphic encryption," in *29th Annual Conference on Neural Information Processing Systems (NIPS)*, 2016.

[26] S. Park, J. Lee, J. H. Cheon, J. Lee, J. Kim, and J. Byun, "Security-preserving support vector machine with fully homomorphic encryption." in *SafeAI@ AAAI*, 2019.

[27] C. Dannen, *Introducing Ethereum and solidity*. Springer, 2017, vol. 1.

[28] "FAA B4UFLY," Oct. 2019, https://goo.gl/EdxuJ9.

[29] C. Gentry et al., "Fully homomorphic encryption using ideal lattices." in *Stoc*, vol. 9, no. 2009, 2009, pp. 169–178.

[30] B. Shrestha, N. Saxena, H. T. T. Truong, and N. Asokan, "Drone to the rescue: Relay-resilient authentication using ambient multi-sensing," in *International Conference on Financial Cryptography and Data Security*. Springer, 2014, pp. 349–364.

[31] Y. Ma, N. Selby, and F. Adib, "Drone relays for battery-free networks," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 335–347.

[32] L. Tang and G. Shao, "Drone remote sensing for forestry research and practices," *Journal of Forestry Research*, vol. 26, no. 4, pp. 791–797, 2015.

[33] J. Wang, Z. Feng, Z. Chen, S. George, M. Bala, P. Pillai, S.-W. Yang, and M. Satyanarayanan, "Bandwidth-efficient live video analytics for drones via edge computing," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 159–173.

[34] H. Fu, S. Abeywickrama, L. Zhang, and C. Yuen, "Low-complexity portable passive drone surveillance via sdr-based signal processing," *IEEE Communications Magazine*, vol. 56, no. 4, pp. 112–118, 2018.

[35] E. Vattapparamban, İ. Güvenç, A. İ. Yurekli, K. Akkaya, and S. Uluağaç, "Drones for smart cities: Issues in cybersecurity, privacy, and public safety," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2016 International*. IEEE, 2016, pp. 216–221.

[36] A. Harrington, "Who controls the drones?[regulation unmanned aircraft]," *Engineering & Technology*, vol. 10, no. 2, pp. 80–83, 2015.

[37] B. Jenkins, "Watching the watchmen: Drone privacy and the need for oversight," *Ky. LJ*, vol. 102, p. 161, 2013.

[38] A. Cavoukian, *Privacy and drones: Unmanned aerial vehicles*. Information and Privacy Commissioner of Ontario, Canada Ontario, Canada, 2012.

[39] M. L. Ward, P. M. Czarnecki, and R. J. Anderson, "Geo-fencing in a wireless location system," Nov. 27 2012, uS Patent 8,320,931.

[40] M. Grainger, R. Mayor, and R. K. Huang, "Beacon-based geofencing," Mar. 12 2013, uS Patent 8,396,485.

[41] X. Shi, C. Yang, W. Xie, C. Liang, Z. Shi, and J. Chen, "Anti-drone system with multiple surveillance technologies: Architecture, implementation, and challenges," *IEEE Communications Magazine*, vol. 56, no. 4, pp. 68–74, 2018.

[42] T. Dasu, Y. Kanza, and D. Srivastava, "Geofences in the sky: herding drones with blockchains and 5g," in *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2018, pp. 73–76.

[43] B. McGillion, T. Dettenborn, T. Nyman, and N. Asokan, "Open-tee–an open virtual trusted execution environment," in *Trustcom/BigDataSE/ISPA, 2015 IEEE*, vol. 1. IEEE, 2015, pp. 400–407.

[44] T. Liu, A. Hojjati, A. Bates, and K. Nahrstedt, "Alidrone: Enabling trustworthy proof-of-alibi for commercial drone compliance," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 841–852.

[45] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010, pp. 24–43.

[46] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Annual Cryptology Conference*. Springer, 2013, pp. 75–92.

[47] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, p. 13, 2014.

[48] S. Halevi and V. Shoup, "Algorithms in helib," in *Annual Cryptology Conference*. Springer, 2014, pp. 554–571.

[49] "Simple encrypted arithmetic library (seal) by microsoft," Jan. 2020, https://www.microsoft.com/en-us/research/project/microsoft-seal/.

[50] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption library," August 2016, https://tfhe.github.io/tfhe/.

[51] A. Prasitsupparote, Y. Watanabe, and J. Shikata, "Implementation and analysis of fully homomorphic encryption in wearable devices," in *The Fourth International Conference on Information Security and Digital Forensics. The Society of Digital Information and Wireless Communications*, 2018, pp. 1–14.

[52] V. Mai and I. Khalil, "Design and implementation of a secure cloud-based billing model for smart meters as an internet of things using homomorphic cryptography," *Future Generation Computer Systems*, vol. 72, pp. 327–338, 2017.

[53] W. Feng and Z. Yan, "Mcs-chain: Decentralized and trustworthy mobile crowdsourcing based on blockchain," *Future Generation Computer Systems*, vol. 95, pp. 649–666, 2019.

[54] Z. Sun, Y. Wang, Z. Cai, T. Liu, X. Tong, and N. Jiang, "A two-stage privacy protection mechanism based on blockchain in mobile crowdsourcing," *International Journal of Intelligent Systems*, 2021.

[55] S. E. Chang, Y.-C. Chen, and M.-F. Lu, "Supply chain re-engineering using blockchain technology: A case of smart contract based tracking process," *Technological Forecasting and Social Change*, vol. 144, pp. 1–11, 2019.

[56] A. Dolgui, D. Ivanov, S. Potryasaev, B. Sokolov, M. Ivanova, and F. Werner, "Blockchain-oriented dynamic modelling of smart contract design and execution in the supply chain," *International Journal of Production Research*, vol. 58, no. 7, pp. 2184–2199, 2020.

[57] S. Wang, D. Li, Y. Zhang, and J. Chen, "Smart contract-based product traceability system in the supply chain scenario," *IEEE Access*, vol. 7, pp. 115 122–115 133, 2019.

[58] A. Ossamah, "Blockchain as a solution to drone cybersecurity," in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*. IEEE, 2020, pp. 1–9.

[59] T. Rana, A. Shankar, M. K. Sultan, R. Patan, and B. Balusamy, "An intelligent approach for uav and drone privacy security using blockchain methodology," in *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*. IEEE, 2019, pp. 162–167.

[60] Y. Wu, H.-N. Dai, H. Wang, and K.-K. R. Choo, "Blockchain-based privacy preservation for 5g-enabled drone communications," *IEEE Network*, vol. 35, no. 1, pp. 50–56, 2021.

[61] M. A. Ferrag and L. Maglaras, "Deliverycoin: An ids and blockchain-based delivery framework for drone-delivered services," *Computers*, vol. 8, no. 3, p. 58, 2019.

[62] M. Dorigo and L. M. Gambardella, "Ant colonies for the travelling salesman problem," *biosystems*, vol. 43, no. 2, pp. 73–81, 1997.

[63] R. Samar and W. A. Kamal, "Optimal path computation for autonomous aerial vehicles," *Cognitive Computation*, vol. 4, no. 4, pp. 515–525, 2012.

[64] B. M. Sathyaraj, L. C. Jain, A. Finn, and S. Drake, "Multiple uavs path planning algorithms: a comparative study," *Fuzzy Optimization and Decision Making*, vol. 7, no. 3, p. 257, 2008.

[65] C. Goerzen, Z. Kong, and B. Mettler, "A survey of motion planning algorithms from the perspective of autonomous uav guidance," *Journal of Intelligent and Robotic Systems*, vol. 57, no. 1-4, p. 65, 2010.

[66] Z. Fu, J. Yu, G. Xie, Y. Chen, and Y. Mao, "A heuristic evolutionary algorithm of uav path planning," *Wireless Communications and Mobile Computing*, vol. 2018, 2018.

[67] A. Ahmadzadeh, J. Keller, G. Pappas, A. Jadbabaie, and V. Kumar, "An optimization-based approach to time-critical cooperative surveillance and coverage with uavs," in *Experimental Robotics*. Springer, 2008, pp. 491–500.

[68] Y. Huang, Q. Yi, and M. Shi, "An improved dijkstra shortest path algorithm," in *Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering*. Atlantis Press, 2013.

[69] Z. He and L. Zhao, "The comparison of four uav path planning algorithms based on geometry search algorithm," in *2017 9th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, vol. 2. IEEE, 2017, pp. 33–36.

[70] J. Daniels, "As Sand Fire rages, feds turn up heat in fight against drones interfering in wildfires," https://goo.gl/NBECb5, July 2016.

[71] B. Coppens, I. Verbauwhede, K. De Bosschere, and B. De Sutter, "Practical mitigations for timing-based side-channel attacks on modern x86 processors," in *Security and Privacy, 2009 30th IEEE Symposium on*. IEEE, 2009, pp. 45–60.

[72] M.-W. Shih, S. Lee, T. Kim, and M. Peinado, "T-sgx: Eradicating controlled-channel attacks against enclave programs," in *ISOC Network and Distributed System Security Symposium*, 2017.

[73] N. Weichbrodt, A. Kurmus, P. Pietzuch, and R. Kapitza, "Asyncshock: Exploiting synchronisation bugs in intel sgx enclaves," in *European Symposium on Research in Computer Security*. Springer, 2016, pp. 440–457.

[74] S. Peterson and P. Faramarzi, "Exclusive: Iran hijacked US drone, says Iranian engineer," https://goo.gl/3gK56T, Dec. 2011.

[75] J. Saarinen, "Students hijack luxury yacht with GPS spoofing," https://goo.gl/BvXi61, July 2013.

[76] K. D. Wesson, D. P. Shepard, J. A. Bhatti, and T. E. Humphreys, "An evaluation of the vestigial signal defense for civil gps anti-spoofing," in *Proceedings of the ION GNSS Meeting*, 2011.

[77] M. L. Psiaki, B. W. O'Hanlon, J. A. Bhatti, D. P. Shepard, and T. E. Humphreys, "Gps spoofing detection via dual-receiver correlation of military signals," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 4, pp. 2250–2267, 2013.

[78] B. W. O'Hanlon, M. L. Psiaki, J. A. Bhatti, D. P. Shepard, and T. E. Humphreys, "Real-time gps spoofing detection via correlation of encrypted signals," *Navigation*, vol. 60, no. 4, pp. 267–278, 2013.

[79] J. Wang, W. Tu, L. C. Hui, S. Yiu, and E. K. Wang, "Detecting time synchronization attacks in cyber-physical systems with machine learning techniques," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 2246–2251.

[80] F. Kaup, P. Gottschling, and D. Hausheer, "Powerpi: Measuring and modeling the power consumption of the raspberry pi," in *Local Computer Networks (LCN), 2014 IEEE 39th Conference on*. IEEE, 2014, pp. 236–243.

[81] H. Chen, K. Laine, and R. Player, "Simple encrypted arithmetic library-seal v2. 1," in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 3–18.

[82] J. Schaad, "Use of the advanced encryption standard (aes) encryption algorithm in cryptographic message syntax (cms)," 2003.

[83] "Raspberry pi 3 model b," May 2017, https://goo.gl/tHUidL.

[84] "The drone pi," Oct. 2019, https://goo.gl/vzqajc.

[85] "scikit-learn, machine learning in python," Jan. 2020, https://scikit-learn.org/stable/.

[86] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[87] M. Radmanesh, M. Kumar, P. H. Guentert, and M. Sarim, "Overview of path-planning and obstacle avoidance algorithms for uavs: a comparative study," *Unmanned systems*, vol. 6, no. 02, pp. 95–118, 2018.

[88] K. Sabe, M. Fukuchi, J.-S. Gutmann, T. Ohashi, K. Kawamoto, and T. Yoshigahara, "Obstacle avoidance and path planning for humanoid robots using stereo vision," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 1. IEEE, 2004, pp. 592–597.

[89] V. Kunchev, L. Jain, V. Ivancevic, and A. Finn, "Path planning and obstacle avoidance for autonomous mobile robots: A review," in *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*. Springer, 2006, pp. 537–544.

[90] "Array of things," Jan. 2020, https://arrayofthings.github.io.

[91] "Chicago data portal," Jan. 2020, https://https://data.cityofchicago.org.

[92] "Gurobi optimization," Jan. 2020, https://www.gurobi.com.

[93] C. Chen, J. Tang, Z. Jin, Y. Yang, and L. Qian, "A path planning algorithm for seeing eye robots based on v-graph," *Mechanical Science and Technology for Aerospace Engineering*, vol. 33, no. 4, pp. 490–495, 2014.